



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática



TFG del Grado en Ingeniería Informática

PLGRAM
Documentación Técnica



Presentado por Víctor Renuncio Tobar
en Universidad de Burgos — 6 de julio de 2016
Tutores: Dr. César Ignacio García Osorio y D. Álgar
Arnaiz González

Índice general

Índice general	I
Índice de figuras	III
Apéndice A Manuales	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	9
Apéndice B Especificación de Requisitos	12
B.1. Introducción	12
B.2. Objetivos generales	12
B.3. Catálogo de requisitos	13
B.4. Especificación de requisitos	14
Apéndice C Especificación de diseño	18
C.1. Introducción	18
C.2. Diseño de datos	19
C.3. Diseño procedimental	24
C.4. Diseño arquitectónico	27
Apéndice D Documentación técnica de programación	33
D.1. Introducción	33
D.2. Estructura de directorios	34
D.3. Manual del programador	34
D.4. Descarga, instalación y ejecución del proyecto	35
D.5. Pruebas del sistema	36
D.6. Métricas	37
Apéndice E Documentación de usuario	39

<i>ÍNDICE GENERAL</i>	II
E.1. Introducción	39
E.2. Requisitos de usuarios	39
E.3. Instalación	39
E.4. Manual del usuario	40
Bibliografía	59

Índice de figuras

B.1. Diagrama principal de casos de uso.[3]	16
C.1. Diagrama de clases: Estructura de las gramáticas.	20
C.2. Diagrama de clases: Estructura de las tablas de análisis.	21
C.3. Diagrama de clases: Estructura del informe del análisis.	22
C.4. Diagrama de clases: Estructura de las clases del análisis.	23
C.5. Diagrama de clases: Estructura del prototipo.	24
C.6. Ejemplo de una estructura de datos.	25
C.7. Esquema general de las plantillas.	26
C.8. Estructura paquete <i>analisis</i> .	27
C.9. Estructura paquete <i>analisis.analisisSintactico</i> .	28
C.10.Estructura paquete <i>analisis.analisisSintactico.ascendente</i> .	28
C.11.Estructura paquete <i>analisis.informe</i> .	29
C.12.Estructura paquete <i>analisis.tabla</i> .	29
C.13.Estructura paquete <i>gramaticas</i> .	30
C.14.Estructura paquete <i>parser</i> .	30
C.15.Estructura paquete <i>prototipo</i> .	31
C.16.Estructura paquete <i>ui</i> .	31
D.1. Icono PLGRAM.	35
D.2. Resultados del análisis de cobertura de pruebas.	36
D.3. Métricas obtenidas con SourceMonitor.	37
D.4. Gráfico de Kiviat obtenido con SourceMonitor.	38
E.1. Icono PLGRAM.	40
E.2. <i>PLGRAM</i> sin argumento gramática.	41
E.3. <i>PLGRAM</i> con argumentos gramática y tipo de análisis.	42
E.4. Aplicación <i>PLGRAM</i> .	43
E.5. Añadir preguntas desde menú Archivo.	44
E.6. Añadir preguntas desde el desplegable.	44

E.7. Panel de preguntas.	45
E.8. Borrar preguntas.	45
E.9. Cambiar el orden de las preguntas.	46
E.10. Cargar gramáticas.	46
E.11. Ejemplo de una gramática cargada en el panel de preguntas.	47
E.12. Ejemplo de cadena.	48
E.13. Ejemplo de la previsualización de un análisis.	49
E.14. Exportar ficheros.	50
E.15. Acerca de.	51
E.16. Estructura del fichero .XML.	52
E.17. Importar fichero .XML en Moodle.	53
E.18. Lista de ficheros .XML en Moodle.	54
E.19. Ejemplo de cuestionario <i>LL</i> resuelto en Moodle.	54
E.20. Ejemplo de cuestionario <i>LR</i> (1) resuelto en Moodle. Conjunto de items	55
E.21. Ejemplo de cuestionario <i>LR</i> (1) resuelto en Moodle. Tabla de AC- CIÓN e IR A	55
E.22. Ejemplo fichero .TEX para un análisis <i>LL</i>	56
E.23. Compilación de un fichero .TEX.	56
E.24. Obtención del cuestionario completo.	57
E.25. Obtención del cuestionario en blanco: descomentar.	57
E.26. Obtención del cuestionario en blanco.	58

Apéndice A

Manuales

A.1. Introducción

En este anexo se detalla el estudio desde el punto de vista temporal y de la viabilidad del proyecto software.

La planificación es una de las tareas más importantes en el desarrollo de un proyecto software y servirá para determinar objetivos, evaluar la viabilidad del proyecto, priorizar actividades...

En la primera parte del anexo se detallará la planificación temporal del proyecto teniendo en cuenta la metodología ágil que se va a utilizar: *Scrum*. En esta fase se determinarán los elementos que forman el *Product Backlog* y la prioridad de cada uno de ellos. Debido a la metodología empleada, no se utilizará el clásico diagrama de *GANTT*. En lugar de esto, se definirá el *Product Backlog* y para el seguimiento se utilizará una herramienta de gestión especializada en metodologías ágiles, *VersionOne*, que permite el seguimiento diario de las tareas por parte del equipo de desarrollo.

En la segunda parte se justificará el desarrollo del proyecto desde diversos puntos de vista: viabilidad técnica, legal, económica, etc.

A.2. Planificación temporal

Esta sección detalla el trabajo realizado en cada *sprint* del proyecto. Tras la primera reunión con el tutor, se decidió que cada *sprint* duraría una semana y se correspondería con una serie de objetivos planteados y unos resultados a conseguir.

A continuación se describe semana por semana dichos objetivos y los resultados que se fueron consiguiendo. Se incluye adicionalmente un *sprint* inicial que transcurre entre la asignación del proyecto y el comienzo oficial.

Sprint 0: hasta el 3 marzo

Esta iteración tiene consideraciones especiales, puesto que desde que se asignaron los proyectos hasta que realmente se comenzó a trabajar semana a semana con el tutor, se realizó una búsqueda de información relacionada con la futura aplicación.

Para ello se leyeron los apuntes de la asignatura *Procesadores del Lenguaje*, se buscó información sobre programas similares y se recabó información sobre los programas que iban a ser de utilidad a la hora de desarrollar el proyecto.

Así, se creó el repositorio en *Github* y se comenzó a realizar la programación temporal utilizando el software online *VersionOne*.

Sprint 1: 3 marzo - 10 marzo

La primera reunión con el tutor fue el 3 de marzo. En ella, se habló sobre cómo se podría distribuir mejor la aplicación que se iba a crear. Se decidió entonces optar por dos soluciones: un prototipo de línea de comandos y una aplicación similar al prototipo con una interfaz gráfica accesible.

Planteados los objetivos generales del proyecto, se empezó a trabajar en el proyecto actual.

Se planteó el primer objetivo, que fue empezar a familiarizarse con las clases de *Burgram*. Para facilitar dicha familiarización con la estructura de clases de *Burgram* y de *PLQuiz*, el tutor me recomendó instalar un plugin que de forma automática generase diagramas *UML* con la arquitectura de las aplicaciones. Este plugin sería: <http://www.objectaid.com/class-diagram>.

Otro de los objetivos planteados fue crear un prototipo de línea de comandos, concretamente la idea sería tener una pequeña aplicación en modo texto que:

1. Tome como argumento el nombre de un archivo con la especificación de una gramática en el formato *bison*.
2. Utilice su contenido para, utilizando las clases adecuadas de *Burgram*, instanciar un objeto *Gramática*.
3. Interrogue el objeto *Gramática* para mostrar el *FIRST* y el *FOLLOW* de los símbolos de la gramática.

Sprint 2: 10 marzo - 17 marzo

El 10 de marzo se realizó una reunión en la que se valoraron los resultados del primer *sprint*. Todos los resultados fueron satisfactorios, se consiguió crear

un prototipo en el cual, introduciendo una gramática en formato *bison*, se obtenía de ella el *FIRST* y el *FOLLOW* por pantalla.

En esta reunión se plantearon nuevos objetivos:

- Generar la tabla de análisis sintáctico predictivo (*TASP*) en el prototipo.
- Instalar $\text{MiK}\text{T}_{\text{E}}\text{X}$.
- Instalar *T_EX*Maker.
- Hacer un uso correcto de los Path.

Sprint 3: 17 marzo - 31 marzo

El 17 de marzo se realizó la siguiente reunión en la que se valoraron los resultados del segundo *sprint*. Todos los resultados fueron satisfactorios. Se consiguió generar la *TASP* y se preparó el prototipo para que los path no diesen problemas en los diferentes sistemas operativos.

Los objetivos que se plantearon en esta reunión fueron:

- Introducir parámetros en el prototipo.
- Investigar sobre diferentes motores de plantillas, como por ejemplo *Mustache*, *Trimou*, *JinJava*, *FreeMaker* o *Shootout*, y decidir cuál era el óptimo para el proyecto.
- Generar una plantilla para crear ficheros .XML utilizables por *MOOD-LE*.
- Generar una plantilla para crear ficheros .TEX.

Se estableció que este *sprint* sería de dos semanas debido a la Semana Santa.

Sprint 4: 31 marzo - 7 abril

El 31 de marzo se realizó la siguiente reunión en la que se valoraron los resultados del tercer *sprint*:

- Se comenzó el tratamiento de datos utilizando una librería llamada *Apache Commons CLI*.
- Se decidió utilizar como motor de plantillas *JinJava*.

- En cuanto a las plantillas, y dado que su generación no había sido tan satisfactoria como se pretendía, se decidió junto con el tutor, centrar el trabajo en la plantilla .XML y dejar la plantilla .TEX para un *sprint* futuro.

En esta reunión se plantearon nuevos objetivos:

- Introducción de *JinJava* para comenzar con la generación de los ficheros deseados.
- Continuar con la plantilla para ficheros .XML.
- Establecer una codificación *UTF* – 8 para todos los ficheros, tanto en las clases del proyecto como en los ficheros generados.

Sprint 5: 7 abril - 14 abril

La siguiente reunión tuvo lugar el 7 de abril y se analizaron los progresos conseguidos en el anterior *sprint*. Se desechó el uso de *JinJava* como motor de plantillas porque no permitía la posibilidad de realizar bucles de repetición. En su lugar, se decidió continuar el trabajo utilizando *Mustache*, ya que cualquier elemento iterable puede ser utilizado. En cuanto a la plantilla para ficheros .XML, no se pudo avanzar lo deseado puesto que aparecieron problemas con *JavaScript* ya que la plataforma *Moodle* de la Universidad de Burgos no permite su uso.

Los objetivos que se plantearon en esta reunión fueron:

- Al tener problemas con el uso de *JavaScript*, se plantearon dos posibles soluciones. Una de ellas era utilizar un motor web que, desde Java, permitiese transformar un documento HTML construido de forma dinámica con *JavaScript*, en un documento HTML estático. La otra solución era cambiar todos los script de *JavaScript* por HTML estático de forma manual. Por tanto, el objetivo era buscar dicho motor web. Si se conseguía encontrar un motor web que transformase los documentos en poco tiempo el objetivo estaría cumplido; sino, se tendría que proceder al cambio de todos los script de forma manual.
- Buscar información sobre el formato de inicio y fin de las preguntas en Moodle.
- Durante la reunión se decidió que las gramáticas estuvieran compuestas por letras mayúsculas y letras minúsculas. Las mayúsculas corresponderían a los no terminales y las minúsculas a los terminales. Se estableció que no se utilizasen símbolos como paréntesis, corchetes, llaves, etc. puesto que interfieren con el funcionamiento de la plantilla. De esta

manera, se planteó como objetivo cambiar las gramáticas y adaptarlas a los nuevos requisitos planteados en esta reunión.

Sprint 6: 14 abril - 21 abril

En la siguiente reunión, que tuvo lugar el 14 de abril, se analizaron los resultados del quinto *sprint*. Debido a la dificultad de encontrar un motor web que permitiese transformar documentos html dinámicos en estáticos, se decidió implantar la segunda solución planteada en la reunión anterior para solucionar el problema que suponía el uso de *JavaScript*. Para ello, se utilizó la propiedad de *Mustache* que permite utilizar cualquier elemento iterable. Los demás objetivos tuvieron resultados satisfactorios.

En esta reunión se plantearon nuevos objetivos:

- Crear respuestas alternativas para los símbolos de preanálisis con cierta lógica conociendo la solución correcta.
- Continuar con la plantilla .XML.

Sprint 7: 21 abril - 28 abril

El 21 de abril se realizó una reunión en la que se valoraron los resultados del sexto *sprint*. Todos los resultados fueron satisfactorios.

Ya que todos los objetivos se iban cumpliendo a tiempo, en esta semana se planteó terminar el código para visualizar las tablas del *FIRST* y del *FOLLOW* en una gramática *LL*.

Sprint 8: 28 abril - 5 mayo

En la siguiente reunión celebrada el 28 de abril se comprobó que los objetivos del séptimo *sprint* se habían conseguido, es decir, el código para visualizar las tablas del *FIRST* y del *FOLLOW* en una gramática *LL* estaba terminado.

En esta reunión se plantearon nuevos objetivos:

- Se estableció cómo debería ser la plantilla y se programó cambiar la plantilla existente en ese momento para que fuera acorde con lo decidido en la reunión.
- Crear la tabla de análisis sintáctico predictivo introduciendo la respuesta correcta. Crear un método para obtener las cadenas de múltiples respuestas. Todo ello para la gramática *LL*.

Sprint 9: 5 mayo - 12 mayo

En la reunión del 5 de mayo se valoraron los resultados obtenidos en el octavo *sprint*. Se comprobó que para la gramática *LL* todo estaba correcto y terminado, menos la tabla de la traza, que se decidió dejar para un futuro *sprint*. Aún así, se decidió meter un parámetro en el prototipo con la cadena para usar posteriormente.

Siguiendo los pasos de la gramática *LL* se programó para este *sprint* comenzar a generar las gramáticas *LR*, *SLR* y *LALR*. Además, se comentó que se debería comenzar a redactar y escribir las partes de la memoria que ya estuvieran definidas hasta ese momento.

Sprint 10: 12 mayo - 19 mayo

En la siguiente reunión, del 12 de mayo, se comprobó que se habían generado las tablas de conjuntos del análisis *LR* pero que todavía faltaban por generar las tablas correspondientes a los otros análisis.

En esta reunión se planteó:

- Crear un método para generar respuestas cortas en Moodle.
- Crear un método para obtener la tabla de *ACCIÓN* e *IR A*.
- Seguir trabajando en los análisis restantes.
- Seguir trabajando en la redacción de la memoria.

Sprint 11: 19 mayo - 26 mayo

El 19 de mayo, en la siguiente reunión se comprobó que se habían generado correctamente las respuestas cortas, así como la tabla de *ACCIÓN* e *IR A* en todas los análisis. Además se presentó un primer avance de la memoria. Se comprobó además que el cambio en el posicionamiento de los conjuntos *LR*, *SLR* y *LALR* era correcto.

El avance de la aplicación por línea de comando era correcto y quedaban pocas cosas por ir terminando, por tanto, se planteó empezar a desarrollar la interfaz gráfica. Además, y dado que los análisis estaban prácticamente finalizados, se programó terminarlos creando la tabla para la traza en todas ellas. Por último, el tutor me facilitó plantillas de memorias de años anteriores para tener modelos a seguir y comenzar con la redacción de los anexos.

Sprint 12: 26 mayo - 2 junio

En la reunión del 26 de mayo se valoraron los resultados del undécimo *sprint* y se comprobó que todos eran satisfactorios.

Aún avanzando a buen ritmo y con la mayoría del proyecto terminado, se decidió en esta reunión presentarlo en la segunda convocatoria para tener tiempo y terminar de pulir aquellas cosas que todavía no estaban del todo correctas.

En esta reunión se comprobó que, en líneas generales, únicamente quedaba por hacer para la finalización del proyecto y, por tanto, serían los objetivos del decimosegundo *sprint*:

- Crear pruebas unitarias para evaluar el funcionamiento de los métodos del prototipo.
- Terminar la interfaz gráfica.
- Retomar y terminar la plantilla .TEX.
- Seguir trabajando en la memoria y en sus anexos.

Sprint 13: 2 junio - 9 junio

En la siguiente reunión, que tuvo lugar el 2 de junio se comprobaron los avances realizados en el anterior *sprint*. Se comprobó que se habían creado correctamente las pruebas unitarias para evaluar el funcionamiento de los métodos del prototipo. Dichas pruebas se han completado con una cobertura suficiente dadas las características del proyecto. Además, se presentaron los avances conseguidos en la plantilla .TEX para los análisis *LL* y se plantearon diferentes alternativas para mejorar la forma de la misma.

Como objetivos para el siguiente *sprint*, se plantearon:

- Continuar con la plantilla .TEX para el resto de análisis e introducir las mejoras planteadas en la reunión.
- Terminar la interfaz gráfica.
- Empezar a pensar en la forma y contenido del póster necesario para la defensa del proyecto.
- Seguir trabajando en la memoria y en sus anexos.

Sprint 14: 9 junio - 16 junio

La siguiente reunión tuvo lugar el 9 de junio y se comprobaron los resultados obtenidos en el anterior *sprint*. Así, se presentaron los avances realizados en la interfaz gráfica y un primer borrador de la memoria. Ambos trabajos fueron satisfactorios, aunque se plantearon mejoras en ambos. También se presentaron los avances en la plantilla .TEX para el resto de análisis que, aún sin

estar finalizados completamente, avanzaban según los criterios planteados en la anterior reunión.

Por tanto, para el siguiente *sprint* se plantearon los siguientes objetivos:

- Terminar la plantilla .TEX.
- Terminar la interfaz gráfica.
- Corregir los errores de la memoria planteados en la reunión y continuar trabajando en los anexos restantes.

Sprint 15: 16 junio - 23 junio

En la siguiente reunión, que tuvo lugar el 16 de junio, se valoraron los avances llevados a cabo en las tareas planteadas para el decimocuarto *sprint*.

Se presentaron las plantillas .TEX para todos los análisis y se comprobó que eran viables. Aún así, se plantearon mejoras en dichas plantillas que se deberían llevar a cabo en el siguiente *sprint*.

Se presentaron además los avances llevados a cabo en la interfaz gráfica. En este caso, aunque los avances que se iban realizando eran satisfactorios, el ritmo de avance no era el deseado y se planteó como objetivo para la siguiente semana terminar definitivamente la interfaz.

Finalmente se entregó la memoria y sus anexos corregidos y mejorados con respecto al borrador anterior para su lectura por los tutores.

Por tanto, para el siguiente *sprint* se planteó como principal objetivo el terminar la interfaz gráfica. También se planteó implementar las mejoras discutidas en las plantillas .TEX.

Sprint 15: 23 junio - 30 junio

La siguiente reunión tuvo lugar el 23 de junio y se valoraron los resultados obtenidos en el *sprint* anterior:

- Con respecto a la plantilla .TEX, se dieron por buenos los resultados obtenidos.
- Se plantearon algunas mejoras en la memoria y anexos presentados en la anterior reunión.
- Con respecto a la interfaz gráfica, se tomaron nuevas decisiones en cuanto al alcance de la misma. Debido a la complejidad de crear una interfaz que funcionara para todos los análisis usando todas las gramáticas, se comprobó que a la hora de exportar los resultados, los datos obtenidos

no eran los correctos en algunos de los casos. Así, se tomó la decisión de finalizar la interfaz gráfica llegando solo a los resultados obtenidos hasta ese momento y se planteó como futuros trabajos relacionados con *PLGRAM*, la mejora de la interfaz gráfica.

Por tanto, el objetivo planteado para el siguiente *sprint* fue continuar con la documentación del proyecto, planteando tener terminada la memoria y sus anexos para la siguiente reunión.

Sprint 16: 30 junio - 7 julio

En este caso, debido a que en Burgos estaba teniendo lugar la celebración de las Fiestas de San Pedro y San Pablo, no hubo una reunión como tal, sino que se envió la memoria y sus anexos a los tutores para su última revisión antes de la entrega final del proyecto.

A.3. Estudio de viabilidad

El análisis de viabilidad es un requisito imprescindible en cualquier organización de un proyecto, sea éste del tipo que sea.

Son muchos los aspectos que intervienen a la hora de valorar la viabilidad de un proyecto y no es recomendable centrarse únicamente en la relación coste-beneficio del mismo, sino que conviene valorar otros aspectos como la funcionalidad, el tamaño del proyecto, el grado de complejidad del mismo, el equipo de trabajo, etc.

En este caso, y dado la finalidad del proyecto actual, un estudio de viabilidad complejo no es necesario, puesto que el proyecto está planteado dentro de un contexto docente y al amparo de la Universidad de Burgos.

De todos modos, se plantean algunos aspectos generales, dividiendo el estudio de viabilidad en los dos criterios que se han considerado más importantes: la viabilidad económica y viabilidad legal.

Viabilidad económica

En este apartado se explica el estudio de costes y la viabilidad económica del proyecto, para poder determinar si es viable económicamente o debe desestimarse.

Estudio de costes

1. Costes de personal

Para el cálculo del coste de personal se debe tener en cuenta al desarrollador y a los dos tutores del proyecto que también deben recibir una retribución por el apoyo prestado.

- Desarrollador: Graduado en Ingeniería Informática que percibirá un salario de 12 €/hora. Considerando las horas invertidas a la semana, el desarrollador supone 360€/semana. La duración del proyecto ha sido de 16 semanas, por lo que el coste del desarrollador será de 5.760 €.
- Tutores: Titulados en Ingeniería Informática que cobrarán 20 €/hora. Calculando que la dedicación media va a ser de 1 horas/semana y teniendo en cuenta que la duración del proyecto es de 16 semanas, se tiene:

$$2 \text{ tutores} * \frac{20 \text{ euros}}{\text{hora}} * \frac{1 \text{ hora}}{\text{semana}} * 16 \text{ semanas} = 640 \text{ €}$$

Sumando el coste del desarrollador y el de los tutores hacen un total de: $5.760 \text{ €} + 640 \text{ €} = 6.400 \text{ €}$.

Al salario bruto hay que añadir el coste de la Seguridad Social (desglosado en contingencias comunes, seguro desempleo y formación profesional) que supone un 30 % aproximadamente sobre el salario bruto, por lo que dicho coste será: $6.400 \text{ €} \times 0,3 = 1.920 \text{ €}$.

El coste total de personal (sumando el coste de la seguridad social) hace un total de $6.400 \text{ €} + 1.920 \text{ €} = 8.320 \text{ €}$.

2. Coste de software

Se ha trabajado con herramientas cuya licencia es gratuita, por lo que el coste de software utilizado es de 0 €.

3. Coste de Hardware Para el desarrollo del proyecto es necesario un ordenador portátil. Se utilizará un modelo que dispone de un procesador Intel core i7, con 8 GB de RAM con un precio en el mercado de 1000 €.

4. Coste de Impresión

- Impresión de la memoria : 25 €.
- Impresión del poster : 20 €.

5. Costes totales

Debido a que los costes de software son 0 €, el coste total del proyecto asciende a 9.365 €.

Análisis de beneficios

El costal del proyecto ha sido estimado en 9365 € por lo que calculando el precio de la licencia en 150 €, el número de licencias a vender para obtener beneficios sería de 63 licencias.

Por tanto, para considerar económicamente viable el proyecto, sería necesario vender 63 licencias de *PLGRAM*. Debido al carácter universitario de la aplicación desarrollada, se puede intuir que dicho objetivo es factible de conseguir debido a la cantidad de centros universitarios del país.

Viabilidad legal

En el uso de bibliotecas de terceros, como por ejemplo *Apache Commons Cli*, todo el software producido por la *ASF* (*Apache Software Foundation*) o cualquiera de sus proyectos, está desarrollado bajo los términos de la licencia *Apache*, es decir, la licencia permite al usuario del software la libertad de usar dicho software para cualquier propósito, para distribuirlo, modificarlo y distribuir versiones modificadas del software, bajo los términos de la licencia; sin necesidad de preocuparse por problemas legales futuros[2].

Especificación de Requisitos

B.1. Introducción

Este anexo tiene como objetivo analizar y documentar las necesidades funcionales mínimas que deben ser soportadas por la aplicación desarrollada.

Otro objetivo básico del presente anexo es la de priorizar cada una de las funcionalidades a implementar. Si se tiene claro cuáles son las tareas críticas, se podrá llevar a cabo un enfoque más preciso en ellas y asignarlas una mayor cantidad de recursos.

El anexo se estructura presentado una lista de objetivos generales que la aplicación debe cumplir. Son los objetivos que el cliente establece mediante un lenguaje neutral y es trabajo del ingeniero su desglose y formalización. A continuación se incluye un catálogo de los requisitos que deberá cumplir la aplicación.

B.2. Objetivos generales

El desarrollo de la aplicación cumple los siguientes objetivos:

1. Construcción de una *GUI* (Interfaz Gráfica de Usuario) que permita visualizar de forma rápida la aplicación desarrollada.
2. Construcción de una *CLI* (Command Line Interface) que permita trabajar de forma rápida y directa con la aplicación desarrollada.
3. Procesado y manipulación de gramáticas mediante procesadores del lenguaje.
4. Obtención del *FIRST* y del *FOLLOW* de una gramática.

5. Obtención de la Tabla de Análisis Sintáctico Predictivo (*TASP*) de una gramática.
6. Obtención de los conjuntos de items $LR(0)$ y $LR(1)$.
7. Obtención de las tablas de análisis sintáctico *ACCIÓN* e *IR A* para análisis $SLR(1)$, $LR(1)$ y $LALR(1)$.
8. Exportación de cuestionarios en formato .XML compatible con la plataforma Moodle.
9. Exportación de cuestionarios en formato L^AT_EX.
10. Creación de un manual de usuario detallando todas las opciones de la aplicación.

B.3. Catálogo de requisitos

El objetivo de este apartado es definir de forma clara, precisa, completa y verificable todas las funcionalidades y restricciones del sistema a construir.

Funciones del producto

Una de las características más importantes de la aplicación que se ha desarrollado es su facilidad de uso. Dado que su empleo está orientado a convertirse en una herramienta docente, debe ser fácil de usar y posibilitar la creación de cuestionarios de manera rápida, sencilla y comprensible.

Como se explica en el *Anexo E: Manual de usuario*, los archivos obtenidos de la aplicación tendrán extensiones .TEX y .XML, tratando de cubrir un mayor abanico de posibilidades de uso de la aplicación y dando una mayor cobertura a la hora de obtener los cuestionarios.

En el caso de los archivos .XML es necesario el uso de la plataforma *Moodle*, que al ser internacional, se puede utilizar en varios idiomas, dándole así una mayor difusión a la aplicación.

Requisitos de usuario

El usuario debe ser capaz de utilizar la aplicación mediante la Interfaz Gráfica o mediante la Consola del sistema.

La ayuda será un punto a tener en cuenta en el caso de utilizar la aplicación mediante la consola del sistema.

En todo momento el usuario podrá consultar el *Manual del Usuario* para orientarse y resolver dudas que le puedan surgir durante el manejo de la misma.

Requisitos del sistema

Los requisitos mínimos del hardware del sistema serán los mismos que para ejecutar java[1].

Los requisitos software necesarios son:

- Sistema operativo *Windows*, *Linux* o *Macintosh*.
- Máquina virtual de java (*JDK8*).
- Consola de sistema operativo capaz de interpretar Unicode, solo para la interfaz en línea de comandos. Si no soportara Unicode, las tildes u otros símbolos podrían ser mal representados.

B.4. Especificación de requisitos

En este apartado se realizará un análisis de los requisitos de diseño de la aplicación. Para ello, se detallarán los requisitos funcionales y no funcionales de la aplicación.

Para representar graficamente los casos de uso se ha utilizado DIA[3].

Requisitos funcionales

La aplicación debe ser capaz de cumplir los siguientes requisitos:

- *RF1: Cargar gramática y análisis.* Permite al usuario seleccionar una gramática disponible en la carpeta *gramaticas* y un tipo de análisis.
- *RF2: Obtener análisis.* Se obtienen los datos del análisis solicitado para la gramática seleccionada.
- *RF3: Crear cadenas.* Se obtienen las cadenas de los datos del análisis.
- *RF4: Obtener fichero .TEX.* Se utiliza la plantilla para obtener el fichero .TEX.
- *RF5: Obtener fichero .XML.* Se utiliza la plantilla para obtener el fichero .XML.

RF1 Cargar gramática y análisis	
Descripción	Permite al usuario seleccionar una gramática y un tipo de análisis.
Precondiciones	Que exista la gramática.
Secuencia normal	Introducir gramática y análisis.
Postcondiciones	
Excepciones	Si no existe la gramática o tiene algún error.
Rendimiento	
Frecuencia	Alto.
Importancia	Alto.
Urgencia	Alto.

RF2 Obtener análisis	
Descripción	Se obtienen los datos del análisis para la gramática seleccionada.
Precondiciones	Tipo de análisis valido.
Secuencia normal	Introducir análisis.
Postcondiciones	
Excepciones	Si no existe el análisis muestra un error.
Rendimiento	
Frecuencia	Alto.
Importancia	Alto.
Urgencia	Alto.

RF3 crear cadenas	
Descripción	Se obtienen las cadenas del análisis para la gramática seleccionada.
Precondiciones	
Secuencia normal	
Postcondiciones	
Excepciones	
Rendimiento	
Frecuencia	Alto.
Importancia	Alto.
Urgencia	Alto.

RF4 Obtener fichero .TEX	
Descripción	Se utiliza la plantilla para obtener el fichero .TEX.
Precondiciones	Que exista la plantilla PlantillaTEX.
Secuencia normal	
Postcondiciones	
Excepciones	
Rendimiento	
Frecuencia	Medio.
Importancia	Alto.
Urgencia	Alto.

RF5 Obtener fichero .XML	
Descripción	Se utiliza la plantilla para obtener el fichero .XML.
Precondiciones	Que exista la plantilla Plantillamoodle.
Secuencia normal	
Postcondiciones	
Excepciones	
Rendimiento	
Frecuencia	Medio.
Importancia	Alto
Urgencia	Alto

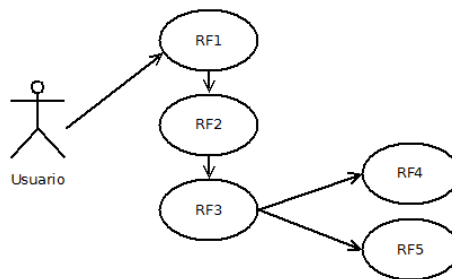


Figura B.1: Diagrama principal de casos de uso.[3]

Requisitos no funcionales

A continuación se describen todos aquellos requisitos no funcionales importantes para considerar en el diseño:

- *RNF1: Eficiencia.* Se debe buscar la eficiencia de la aplicación para que pueda ser usada con gramáticas complejas.
- *RNF2: Extensibilidad.* Se pueden definir nuevas capacidades en la aplicación mediante su extensión a otro tipo de ficheros.
- *RNF3: Práctica.* Al ser una aplicación docente, un requisito es que sea útil y práctica para enseñar/evaluar

Especificación de diseño

C.1. Introducción

En este anexo se detallan todos los aspectos del diseño basándose en el análisis realizado en el *Apéndice B*, buscando una solución a medida para los problemas detectados.

La frontera entre la finalización del análisis y el comienzo del diseño es difusa ya que el modelo evoluciona y se refina en cada paso.

El diseño es fundamental para el correcto desarrollo de un proyecto software porque facilita la estructuración modular, identificando cada elemento del programa y facilitando su uso.

En el diseño del proyecto se pueden diferenciar varias fases:

1. Diseño del tipo abstracto de datos para instanciar los objetos.
2. Diseño de los algoritmos para la obtención del resultado deseado.
3. Diseño de las plantillas para generar los ficheros .TEX y .XML.
4. Diseño de la interfaz de usuario, que se desarrolla de dos formas diferentes:
 - Interfaz en línea de comandos(*CLI*).
 - Interfaz Gráfica de usuario(*GUI*).

C.2. Diseño de datos

El objetivo de este capítulo es seleccionar las representaciones lógicas de los datos (estructuras de datos) que se han identificado en la fase de análisis.

Para la representación se utilizarán diagramas de clases que permiten modelar el diseño de la aplicación.

Se ha intentado conseguir la mayor claridad posible a la hora de representar las clases a generar. Debido a la gran cantidad de clases mostradas, se ha optado por no representar todas, intentando agrupar por funcionalidad y representando tan sólo aquellas que aporten información y valor al diagrama en el que se representan.

Para facilitar la legibilidad de las mismas, se ha optado por no mostrar algunos de los atributos y métodos de algunas de las clases.

Los diagramas de clases del proyecto actual están basados en los obtenidos del programa *BURGRAM*. Los diagramas de *PLGRAM* son una modificación de los anteriores de manera que el trabajo a desarrollar fuera más sencillo y eficaz. A continuación, se muestran algunos de ellos y una pequeña explicación de los mismos.

Estructura de las gramáticas

El caso de la Estructura de las gramáticas, se muestra en la Figura C.1. En ella, se muestran las relaciones y herencias entre las clases. Se ha utilizado Objectaid[4].

Puede observarse que la *Gramática* está compuesta por un *Vector de Producciones*, que a su vez está compuesto por una o varias *Producciones*. Dichas *Producciones* está compuestas por un *Vector de Símbolos*, formado por *Símbolos* o un *No Terminal*. Los *Símbolos* pueden ser *Terminales*, *No Terminales* o *Nulos*.

A su vez, la gramática tiene un *First* y un *Follow*, que pueden estar compuestos por un *Vector Símbolos*.

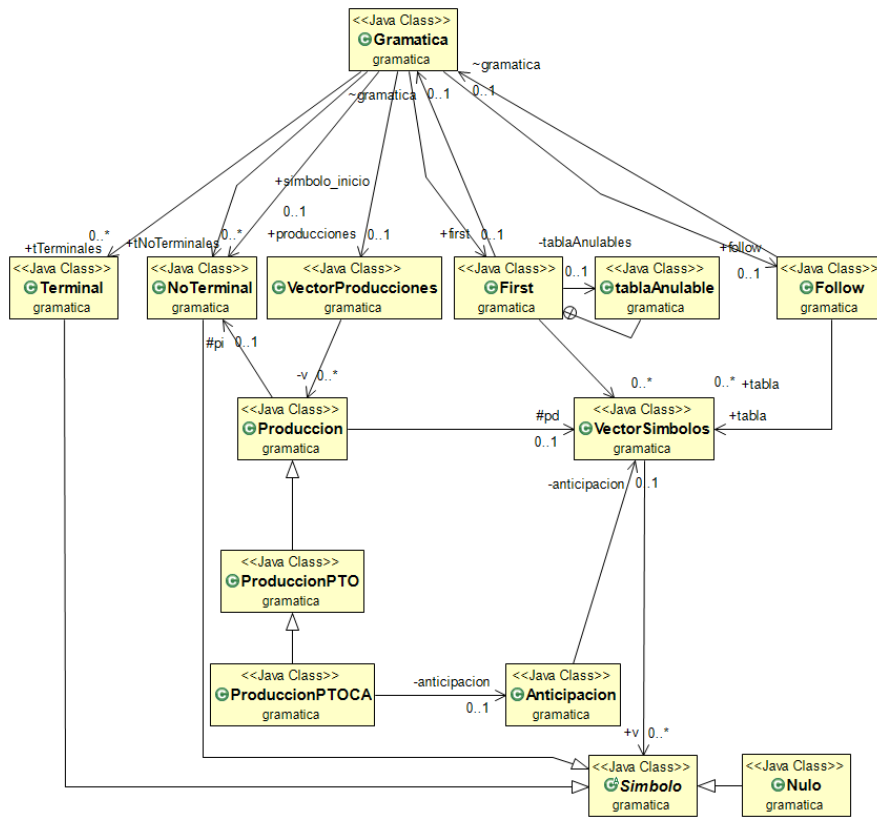


Figura C.1: Diagrama de clases: Estructura de las gramáticas.

Estructura de las tablas de análisis

En la Figura C.2 se muestra la Estructura de las tablas de análisis. Puede observarse que las clases *TablaDescendente* y *TablaAscendente* son subclases de la clase *Tabla* y, por lo tanto, heredan sus métodos.

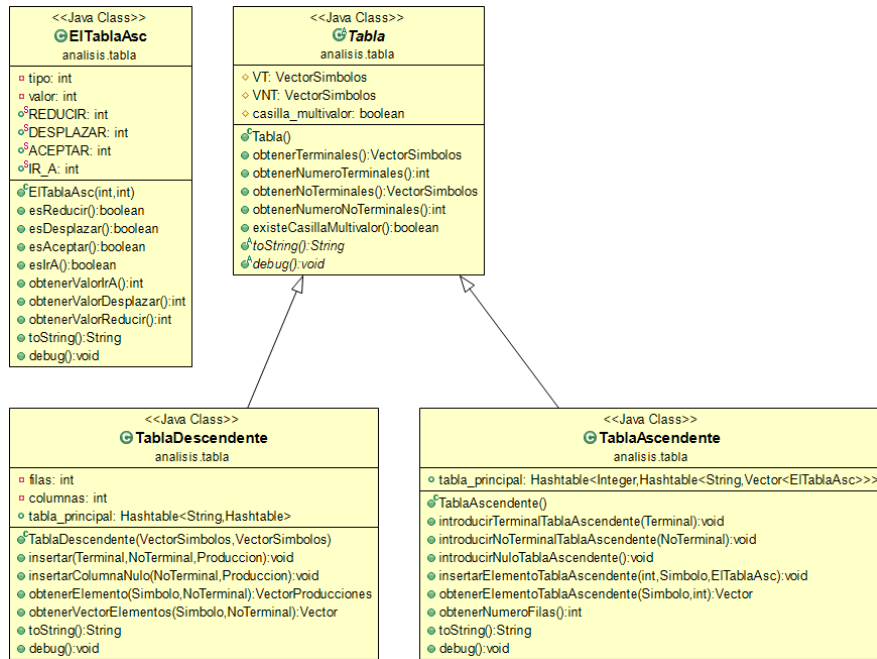


Figura C.2: Diagrama de clases: Estructura de las tablas de análisis.

Estructura del informe del análisis

En este Diagrama de clases, se observa que las clases *InformeAscendente* e *InformeDescendente* heredan de la clase *Informe*, y por tanto, heredan sus métodos.

Existe también una clase interna de la clase *Informe*, denominada *Plantilla*.

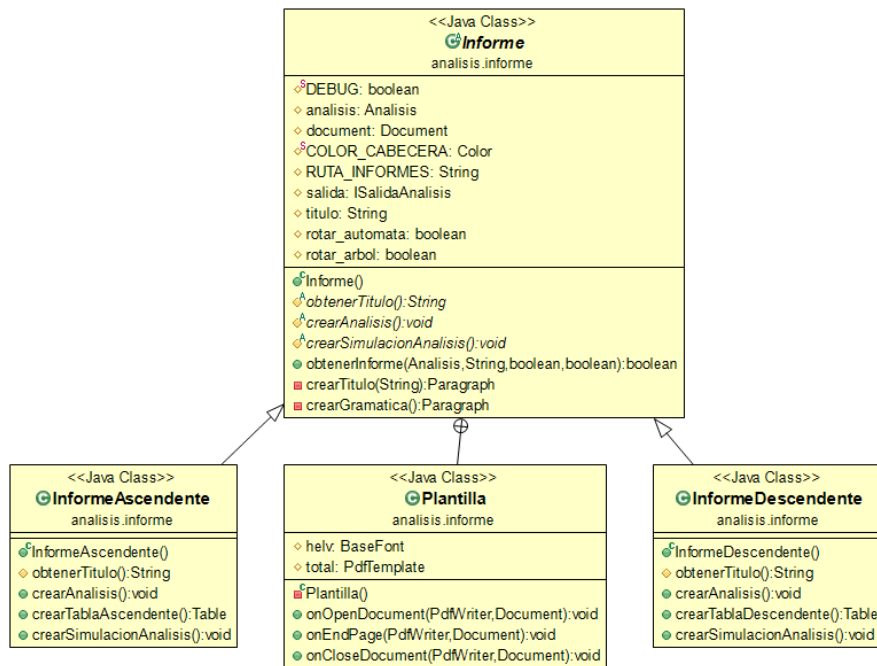


Figura C.3: Diagrama de clases: Estructura del informe del análisis.

Estructura de las clases del análisis

En el Diagrama de clases mostrado en la Figura C.4 se tiene una clase abstracta *Analisis*, de la cual heredan las clases *AnalisisLL1* y *AnalisisSLR1*. La clase *AnalisisSLR1* tiene como subclases las clases *AnalisisLR1* y *AnalisisLALR1*.

Además, se puede observar en el Diagrama la interfaz *ISalidaAnalisis* que aporta un conjunto de métodos comunes para todas las clases posteriores, en este caso, las comentadas anteriormente.

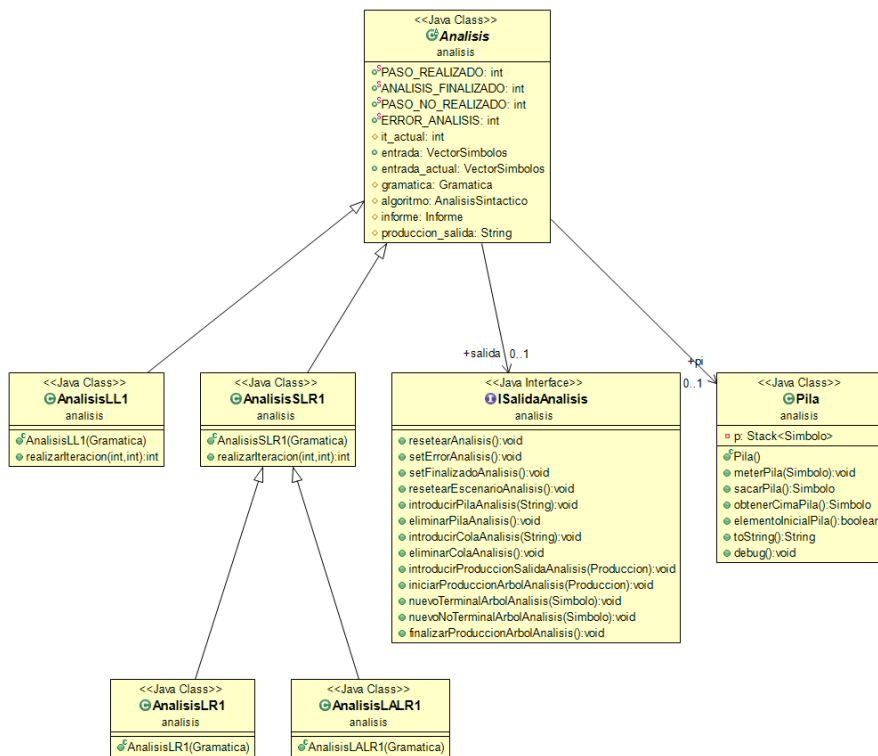


Figura C.4: Diagrama de clases: Estructura de las clases del análisis.

Estructura del prototipo

La clase *Prototipo* está compuesta por diferentes métodos que permiten crear distintas cadenas dependiendo si se quiere un fichero .XML o .TEX.

La clase *Prototipo* tiene diferentes clases internas que permiten guardar la estructura de datos usada en la plantilla. Estas clases internas son las que se muestran en la parte baja del Diagrama de la Figura C.5.

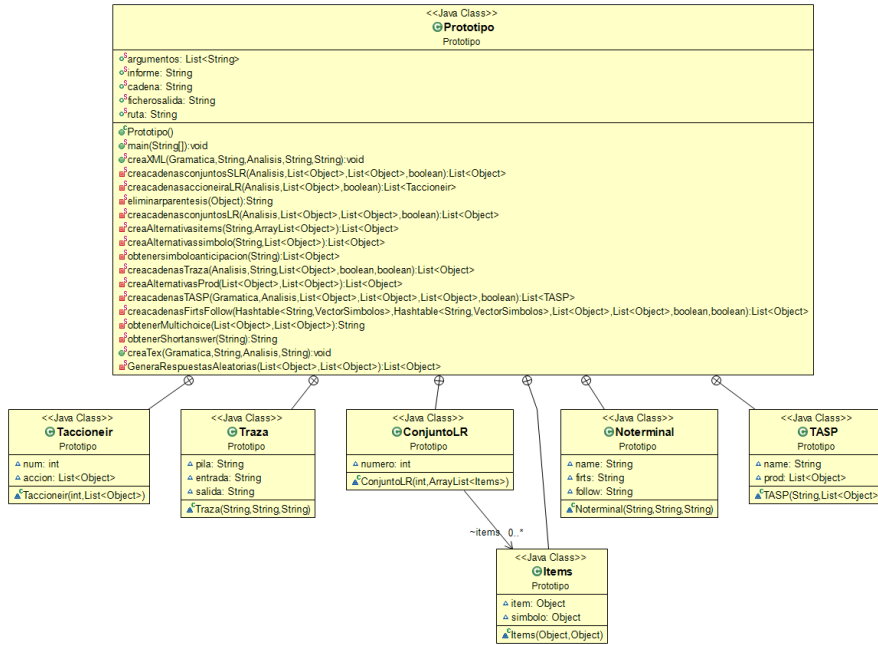


Figura C.5: Diagrama de clases: Estructura del prototipo.

C.3. Diseño procedimental

El proyecto desarrollado, no ha requerido casos de uso complejos, sino que ha sido un tratamiento de cadenas y un almacenamiento de datos en clases con una estructura especial. Estas clases son las clases internas que se han comentado en el Diagrama de clases: estructura del prototipo(ver C.5).

Todos los datos de la plantilla están guardados en un diccionario (*HashMap*) compuesto por claves y valores. Las claves son los nombres que va a utilizar la plantilla. Los valores pueden ser cadenas de caracteres o listas de estructuras de datos.

El diseño procedimental general de la aplicación es bastante sencillo. Básicamente se necesitan dos elementos, una plantilla y unos datos, que como se

ha comentado anteriormente se encuentran en un *HashMap*. Bastaría con compilar la plantilla con dichos datos para obtener el fichero final deseado. Este simple procedimiento se ejemplifica a continuación:

1. Plantilla

```
{{#FirstFollow}}
Nombre: {{name}} first: {{first}} follow: {{follow}}.
{{/FirstFollow}}
```

2. Estructura de datos

Name	Value
▼ FirstFollowSin	ArrayList<E> (id=59)
▼ ▲ elementData	Object[10] (id=71)
▼ ▲ [0]	Prototipo\$Noterminal (id=72)
> ▲ first	"begin" (id=83)
> ▲ follow	"end" (id=84)
> ▲ name	"A" (id=65)
▼ ▲ [1]	Prototipo\$Noterminal (id=73)
> ▲ first	"tipo,id" (id=81)
> ▲ follow	"\$" (id=82)
> ▲ name	"S" (id=66)
▼ ▲ [2]	Prototipo\$Noterminal (id=74)
> ▲ first	"codigo" (id=79)
> ▲ follow	"end" (id=80)
> ▲ name	"C" (id=67)
▼ ▲ [3]	Prototipo\$Noterminal (id=75)
> ▲ first	"tipo,id" (id=77)
> ▲ follow	"begin" (id=78)
> ▲ name	"B" (id=68)

Figura C.6: Ejemplo de una estructura de datos.

3. Resultado obtenido de la compilación de la plantilla anterior con la estructura de datos mostrada.

```
Nombre: A first: begin follow: end.
Nombre: S first: tipo,id follow: $.
Nombre: C first: codigo follow: end.
Nombre: B first: tipo,id follow: begin.
```

Evidentemente, el proyecto es más complejo de lo mostrado en el ejemplo anterior. La dificultad del mismo radica en dos pilares fundamentales:

- Creación de las estructuras de datos. El principal problema fue que las estructuras de datos eran variables según la gramática que se introducía y por tanto debían ser dinámicas. Este problema se solucionó creando las clases internas ya comentadas.
- Diseño de la plantilla. En un principio se planteó crear una plantilla genérica que sirviera para la obtención de los dos tipos de fichero. Aunque actualmente, después del desarrollo de todo el proyecto, es algo que sí podría llevar a cabo, en el momento de empezar con el diseño de las plantillas era algo que se antojaba bastante complejo. Por tanto, se optó por realizar dos diseños de plantillas, uno para cada tipo de fichero. Además, cada tipo de plantilla tenía que servir para cuatro tipos diferentes de análisis, que a su vez están formados por distintos elementos. Este complejo entramado es lo que complicó el diseño de la plantilla. Se tuvo que crear una plantilla que recopilara todos los siguientes datos:

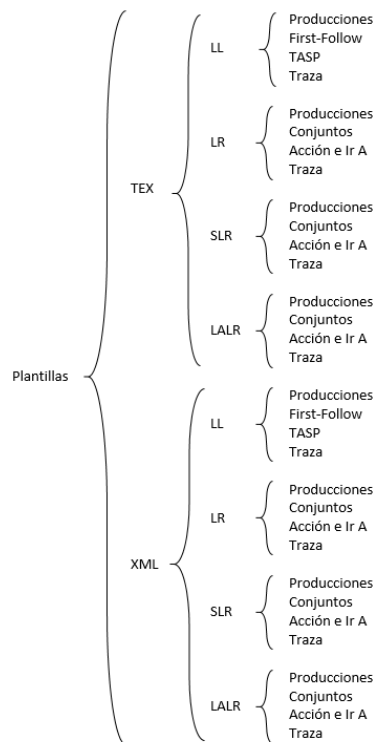


Figura C.7: Esquema general de las plantillas.

C.4. Diseño arquitectónico

El diseño arquitectónico que se ha seguido ha sido similar al del programa *PLQUIZ*.

En *PLGRAM* se han generado dos interfaces de usuario independientes:

- Interfaz gráfica de usuario.
- Interfaz en línea de comandos.

A continuación se detallan los paquetes principales que forman la estructura de la aplicación:

- *src*: Contiene las clases de la aplicación y las plantillas que se utilizan en la misma.
- *test*: Contiene las pruebas unitarias para conseguir un correcto funcionamiento de la aplicación.

src

Dentro del paquete *src* se encuentran los siguientes paquetes:

- *src.analisis*: sirve para guardar las clases donde se representan todas las operaciones de los análisis sintácticos.

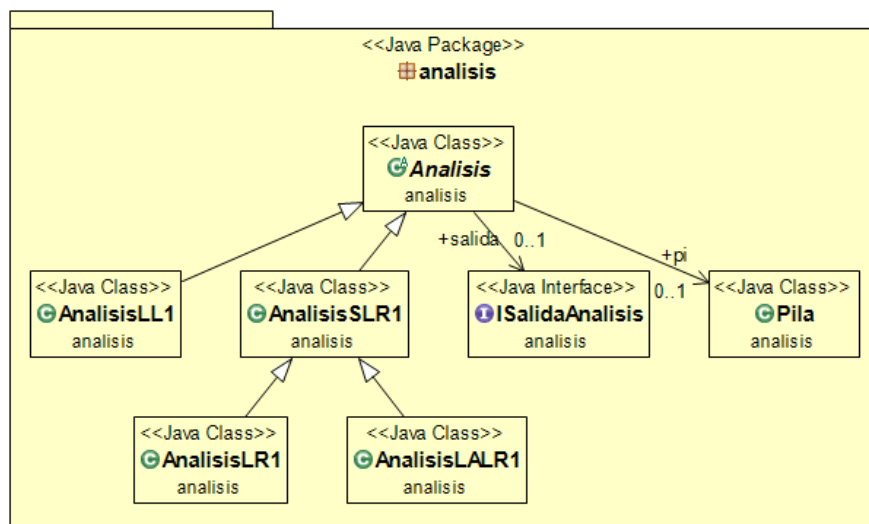


Figura C.8: Estructura paquete *analisis*.

- *src.analisis.analisisSintactico*: Sirve para guardar las clases de los diferentes tipos de análisis *LL*, *SLR*, *LR* y *LALR*.

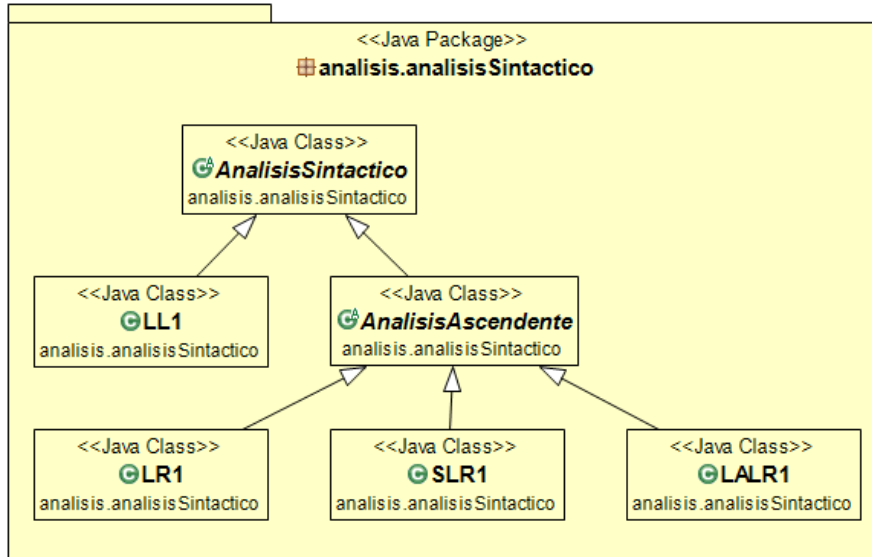


Figura C.9: Estructura paquete *analisis.analisisSintactico*.

- *src.analisis.analisisSintactico.ascendente*: Sirve para guardar las clases que representan a un autómata.

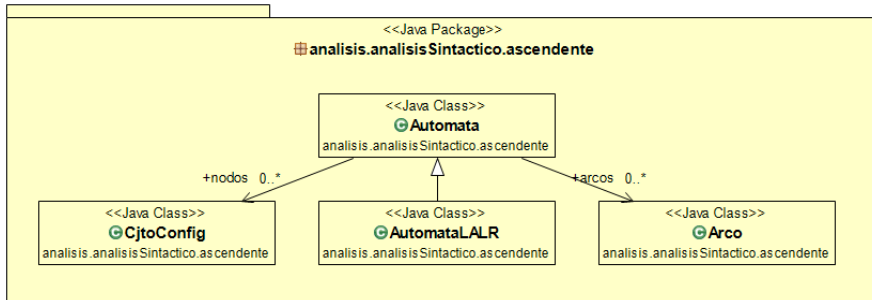
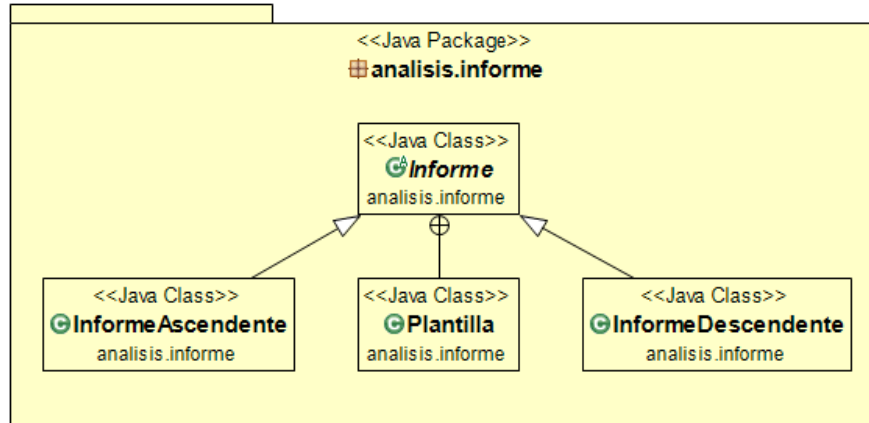
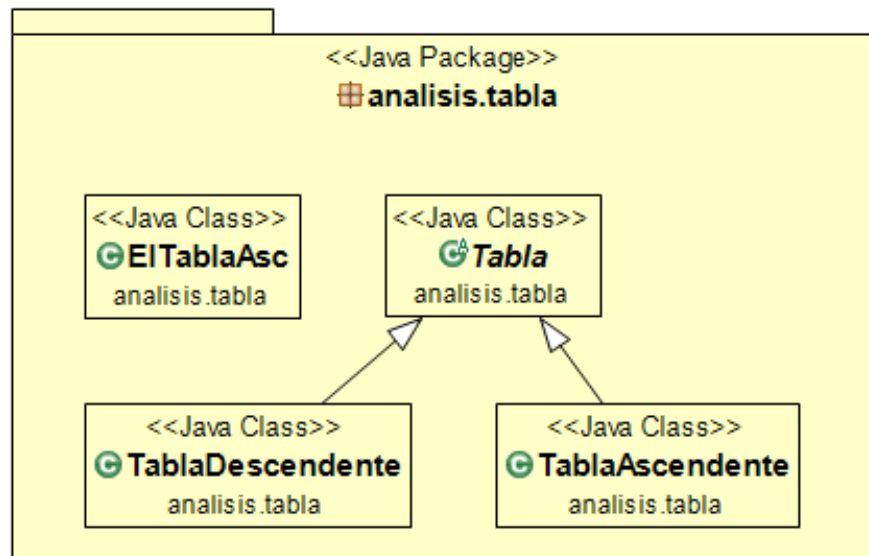


Figura C.10: Estructura paquete *analisis.analisisSintactico.ascendente*.

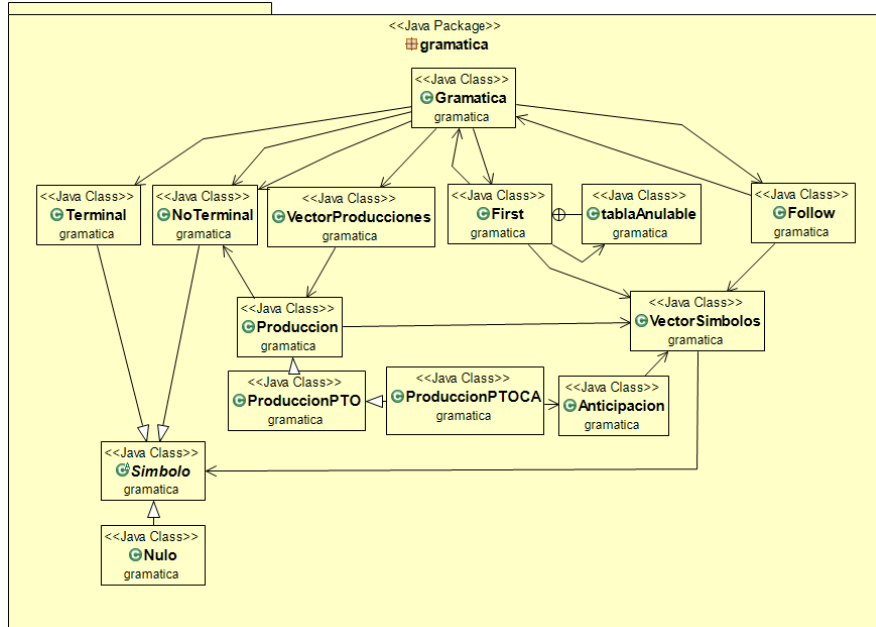
- *src.analisis.informe*: Sirve para guardar las clases donde se representan las diferentes operaciones de los informes generados.

Figura C.11: Estructura paquete *analisis.informe*.

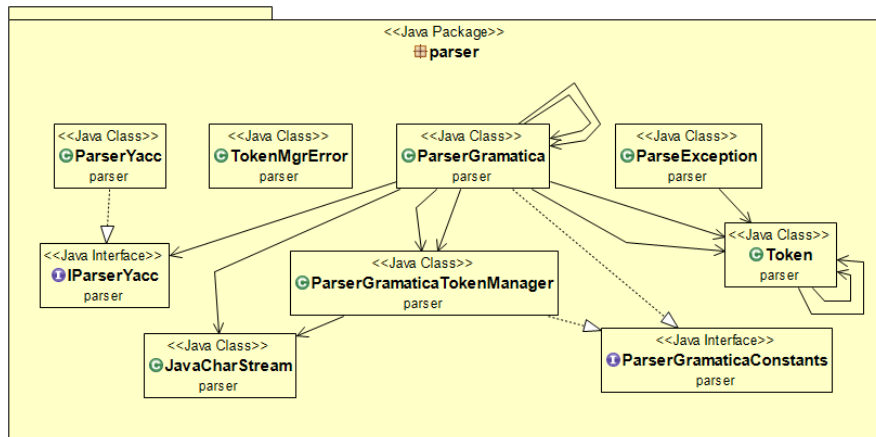
- *src.analisis.tabla*: Sirve para guardar las clases donde se representan todas las operaciones de las tablas del análisis sintáctico.

Figura C.12: Estructura paquete *analisis.tabla*.

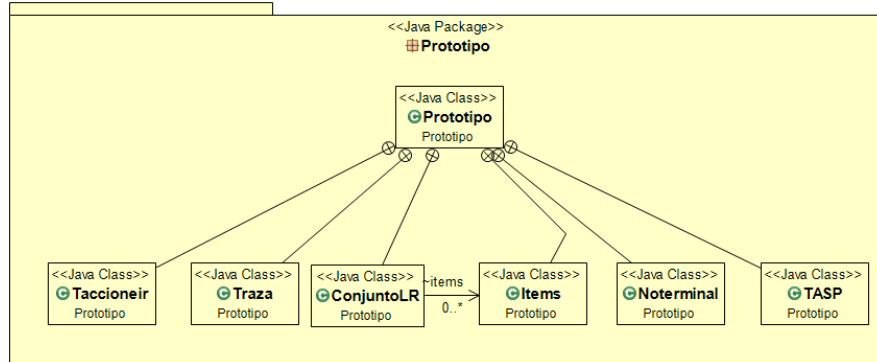
- *src.gramaticas*: Sirve para guardar todas las clases donde se representan todas las operaciones de las gramáticas.

Figura C.13: Estructura paquete *gramaticas*.

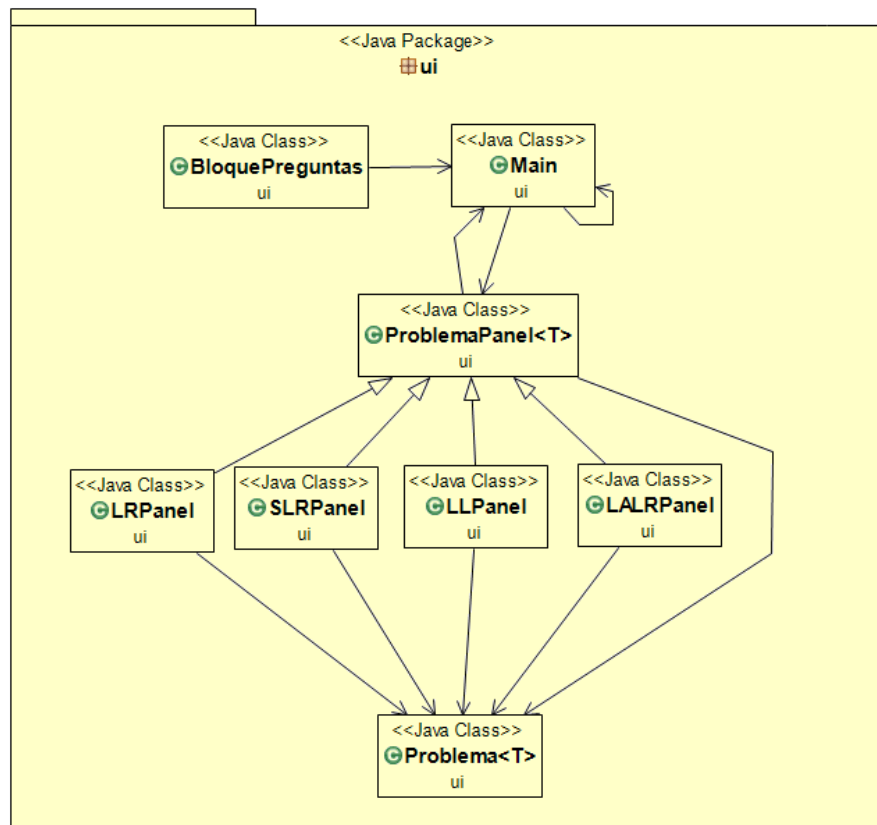
- *src.parser*: Sirve para guardar todas las clases donde se representan todas las operaciones del parser.

Figura C.14: Estructura paquete *parser*.

- *src.prototipo*: Sirve para guardar la clase prototipo.

Figura C.15: Estructura paquete *prototipo*.

- *src.ui*: Sirve para guardar las clases de la interfaz gráfica.

Figura C.16: Estructura paquete *ui*.

test

El paquete *test* está formado únicamente por el paquete por defecto (*default package*) que contiene los test unitarios.

Documentación técnica de programación

D.1. Introducción

En este capítulo se detallan los aspectos más relevantes de la implementación del diseño expuesto en el apéndice C. Además se detallarán las convenciones de estilo y codificación utilizadas durante la fase de programación.

El desarrollo se ha realizado en la versión 8 de *Java*. Para la implementación del núcleo de la aplicación se ha utilizado *Eclipse* como entorno integrado de desarrollo. Para el desarrollo de las interfaces de usuario se ha empleado *Swing* y para el control de versiones se ha utilizado *VersionOne*.

El motivo de escoger *Eclipse* fue que ya se había utilizado en proyectos anteriores, conociendo así sus puntos fuertes y débiles. La potencia de esta herramienta facilita las tareas de programación, sobre todo a la hora de refactorizar o depurar el código fuente. Otra de las ventajas que ofrece, es la cantidad de plugins disponibles: control de tareas, control de versiones, pruebas, etc.

Como en todo proyecto software, era necesario disponer de un control de versiones potente y versátil para controlar los fuentes desarrollados (*Github*) y un control de tareas. Para esta tarea se barajaron diversos programas: *Subversion*, *GIT* y *Plastic SCM*, entre otros. Se escogió *VersionOne*. La licencia gratuita de *VersionOne* caducaba a los 30 días de comenzar a usarlo, borrando todos los datos almacenados en ese momento. Por ello, cuando el proyecto estaba ya muy avanzado, se perdió toda la información que se había ido almacenando en *VersionOne* sin que hubiera ninguna posibilidad de recuperarla¹.

¹<https://www53.v1host.com/Ubu44/Default.aspx?menu=MyHomeEnterpriseGettingStartedPage>

Para la realización de las pruebas unitarias se ha optado por *JUnit*. Para las pruebas de cobertura se ha utilizado un plugin de *Eclipse* llamado *EclEmma* por la calidad en el desempeño de sus tareas, además de ser un plugin ya conocido al haberlo utilizado durante la carrera.

Para la obtención de métricas se ha utilizado *SourceMonitor* por su facilidad de uso y porque, entre otras cosas, aportaba las métricas necesarias para los informes.

D.2. Estructura de directorios

La estructura de directorios que se ha seguido en el soporte físico distribuido junto a la documentación, es la siguiente:

- PLGRAM: Este directorio contiene la aplicación desarrollada, incluyendo los fuentes, las plantillas, los ejecutables y las pruebas. Además, incluye el *build.xml* para *Ant*.
- Memoria: Este directorio contiene la memoria del trabajo.
- Documentación técnica: Este directorio contiene la documentación que se está leyendo en este momento.

D.3. Manual del programador

El manual del programador se encuentra dividido en dos secciones: en la primera, aparecen detalladas las herramientas utilizadas durante el desarrollo y, en la segunda, el fichero de configuración de *Ant* explicando cada uno de los objetivos (o *target*).

Herramientas utilizadas

Las herramientas utilizadas han sido cuidadosamente explicadas en la memoria, en el capítulo 4 Técnicas y herramientas, sección 2 Herramientas.

Configuración de la construcción

Para la construcción de los ejecutables, pruebas, métricas y demás objetos se ha utilizado *Ant*. Esta herramienta sería el equivalente en *Java* al *Make* de *C*.

Ant utiliza ficheros de configuración escritos en XML. Para el proyecto, ha sido generado el fichero *build.xml*, que se encuentra en el directorio PLGRAM.

D.4. Descarga, instalación y ejecución del proyecto

Todo el código de la aplicación está detalladamente comentado para facilitar a un futuro programador la ampliación y programación que se desee realizar.

Descarga

El proyecto puede obtenerse directamente o a través de su repositorio de *Github*², que ofrece dos opciones:

- Puede descargarse como un fichero comprimido mediante la opción Download ZIP.
- Puede descargarse utilizando uno de los clientes propietarios de Github con la opción Clone in Desktop.

Instalación y ejecución

La aplicación se distribuye de dos formas diferentes: *PLGRAM* y *PLGRAMlineCommand*.

- Para ejecutar *PLGRAM* se dispone de un fichero jar ejecutable.

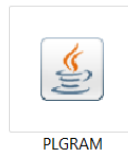


Figura D.1: Icono PLGRAM.

- Para ejecutar *PLGRAMlineCommand* se debe acceder mediante la línea de comandos a la carpeta que contiene el fichero.

Esta carpeta debe contener también una carpeta llamada *gramaticas* donde estarán las gramáticas guardadas. Además, también estarán en ese directorio las plantillas necesarias (*plantillaTEX* y *plantillamoodle*).

²<https://github.com/vrt0004/Trabajo>

D.5. Pruebas del sistema

La aplicación utiliza JUnit 4 para realizar sus pruebas, y cuenta con un total de 27 pruebas. Las pruebas excluyen las clases autogeneradas (como por ejemplo las pertenecientes al paquete parser, generadas por JavaCC). Las herramientas aplicadas indican una cobertura general superior al 55 %, con más del 90 % en las clases más relevantes.

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
PLGRAM	57,6 %	11.894	8.755	20.649
src	56,9 %	11.114	8.405	19.519
> analisis.informe	3,3 %	50	1.443	1.493
> ui	20,2 %	668	2.631	3.299
> analisis	30,9 %	216	482	698
> parser	46,7 %	2.632	3.008	5.640
> analisis.analisisSintactico.ascendi	80,7 %	367	88	455
> analisis.tabla	82,7 %	661	138	799
> gramatica	89,7 %	1.999	230	2.229
> ProduccionPTOCA.java	62,1 %	36	22	58
> Simbolo.java	81,5 %	22	5	27
> VectorProducciones.java	81,8 %	117	26	143
> First.java	83,6 %	595	117	712
> Anticipacion.java	85,1 %	63	11	74
> Nulo.java	92,0 %	23	2	25
> Produccion.java	92,2 %	106	9	115
> NoTerminal.java	93,5 %	29	2	31
> Terminal.java	93,5 %	29	2	31
> Follow.java	95,2 %	279	14	293
> VectorSimbolos.java	95,8 %	205	9	214
> ProduccionPTO.java	96,3 %	105	4	109
> Gramatica.java	98,2 %	390	7	397
> Prototipo	89,9 %	2.639	296	2.935
> Prototipo.java	89,9 %	2.639	296	2.935
> analisis.analisisSintactico	95,5 %	1.882	89	1.971
> AnalisisSintactico.java	83,3 %	15	3	18
> LL1.java	93,8 %	137	9	146
> LR1.java	95,2 %	579	29	608
> LALR1.java	95,3 %	590	29	619
> SLR1.java	96,3 %	490	19	509
> AnalisisAscendente.java	100,0 %	71	0	71
> test	69,0 %	780	350	1.130

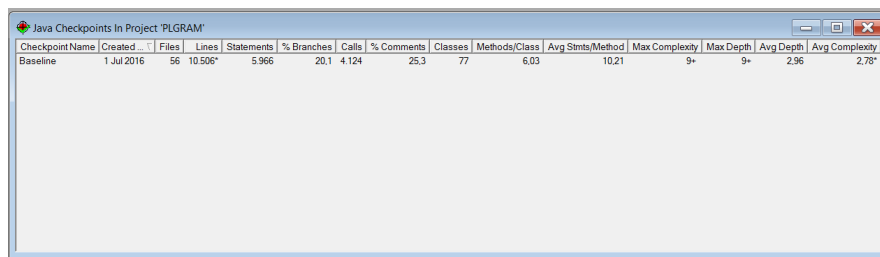
Figura D.2: Resultados del análisis de cobertura de pruebas.

D.6. Métricas

Para las distintas métricas se ha utilizado *SourceMonitor*.

El resultado del análisis obtenido con *SourceMonitor* se puede observar en la tabla siguiente. Se muestra también un gráfico de Kiviat obtenido con *SourceMonitor*.

En ambos, aparecen los umbrales definidos por la Universidad de Burgos. Los valores que no se encuentran dentro del umbral establecido han sido resaltados en negrita.



Checkpoint Name	Created	Files	Lines	Statements	% Branches	Calls	% Comments	Classes	Methods/Class	Avg Strmts/Method	Max Complexity	Max Depth	Avg Depth	Avg Complexity
Baseline	1 Jul 2016	56	10.506*	5.966	20,1	4.124	25,3	77	6,03	10,21	9+	9+	2,96	2,78*

Figura D.3: Métricas obtenidas con SourceMonitor.

Métrica	Resultado	Mín. UBU	Máx. UBU
Líneas de código	10.506		
Número de sentencias	5.966		
Porcentaje de sentencias condicionales	20,1		
Número de llamadas a métodos	4.124		
Porcentaje de líneas de comentarios	25,3	8	20
Número de clases e interfaces	77		
Número de métodos por clase	6,03	4	16
Media de sentencias por método	10,21	6	12
Complejidad máxima	9+	2	8
Media de complejidad	2,78	2	4
Máxima profundidad en bloques	9+	3	7
Media de profundidad de bloques	2,96	1	2,2

Aunque la complejidad máxima y la profundidad máxima se salen de los valores establecidos, la media de ambas métricas se encuentra en el umbral o muy cerca de estarlo.

A continuación se muestra un diagrama de kiviatt de la aplicación PLGRAM.

Los diagramas de Kiviatt son unos gráficos circulares en cuyos ejes radiales se representan diferentes índices de prestaciones. Las intersecciones entre los radios y la circunferencia representan los valores máximos que pueden alcanzar las variables representadas en los mismos. Aunque en principio, el número de ejes que puede tener un gráfico de este tipo es arbitrario y depende de los datos que se van a representar, se suelen seguir unos convenios de representación.

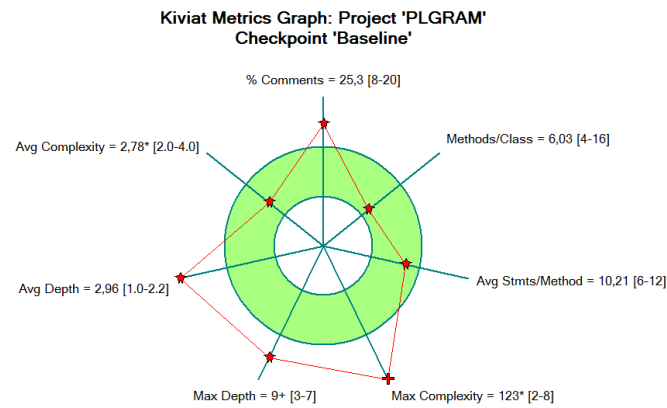


Figura D.4: Gráfico de Kiviatt obtenido con SourceMonitor.

Documentación de usuario

E.1. Introducción

En esta sección se explica cómo llevar a cabo la instalación de las herramientas implementadas en el presente proyecto y los requisitos del sistema.

E.2. Requisitos de usuarios

Los requisitos mínimos del hardware del sistema serán los mismos que para ejecutar java[1]

Los requisitos software necesarios son:

- Sistema operativo *Windows*, *Linux* o *Macintosh*.
- Máquina virtual de java (*JDK8*).
- Consola de sistema operativo capaz de interpretar Unicode, solo para la interfaz en línea de comandos. Si no soportara Unicode, las tildes u otros símbolos podrían ser mal representados.

E.3. Instalación

La aplicación se distribuye de dos formas diferentes: *PLGRAM* y *PLGRAMlineCommand*:

- Para ejecutar *PLGRAM* se dispone de un fichero jar ejecutable.

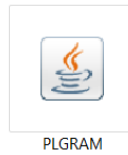


Figura E.1: Icono PLGRAM.

- Para ejecutar *PLGRAMlineCommand* se debe acceder mediante la línea de comandos a la carpeta que contiene el fichero.

Esta carpeta debe contener también una carpeta llamada *gramaticas* donde estarán las gramáticas guardadas. Además, también estarán en ese directorio las plantillas necesarias (*plantillaTEX* y *plantillamoodle*).

E.4. Manual del usuario

Aplicación línea de comandos

Para ejecutar la aplicación se debe escribir el siguiente comando:

```
java -jar PLGRAMlineCommand.jar -g -t [-i] [-ca] [-o]
```

Los diferentes argumentos son:

- *-g* Nombre de la gramática a analizar (que debe estar en la carpeta *gramaticas* mencionada anteriormente).
- *-t* Tipo de análisis a realizar [*LL*, *LR*, *SLR*, *LALR*].
- *-i* Extensión del informe [*TEX*, *XML*, *ALL*].
- *-ca* Cadena a analizar.
- *-o* Nombre del fichero de salida.

La gramática *-g* y el tipo de análisis a realizar *-t* son obligatorios. La gramática debe estar en la carpeta *gramaticas*.

El parámetro *-t* tiene diferentes opciones:

- *LL* Para realizar un análisis *LL*.
- *LR* Para realizar un análisis *LR*.
- *SLR* Para realizar un análisis *SLR*.

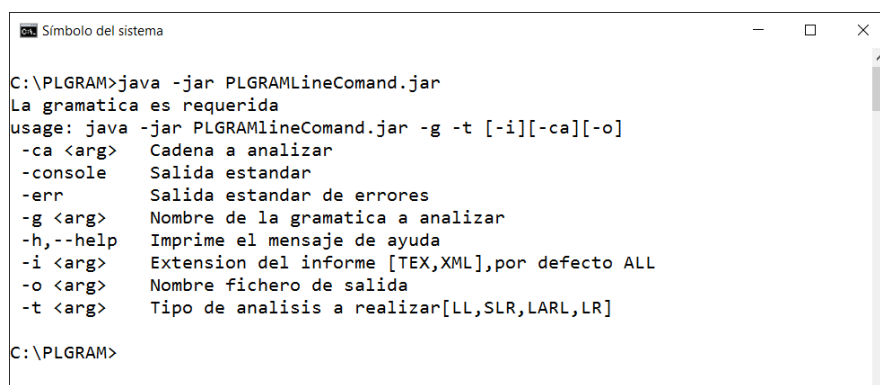
- *LALR* Para realizar un análisis *LALR*.

Si no se introduce alguno de los parámetros obligatorios (*-g* , *-t*) o el parámetro es incorrecto se indicará que falta el parámetro.

Así, si se introduce la siguiente línea de código

```
java -jar PLGRAMlineCommand.jar
```

La aplicación requerirá la gramática, tal y como se muestra a continuación:



```

C:\PLGRAM>java -jar PLGRAMlineComand.jar
La gramatica es requerida
usage: java -jar PLGRAMlineComand.jar -g -t [-i][-ca][-o]
       -ca <arg>  Cadena a analizar
       -console   Salida estandar
       -err       Salida estandar de errores
       -g <arg>   Nombre de la gramatica a analizar
       -h,--help  Imprime el mensaje de ayuda
       -i <arg>   Extension del informe [TEX,XML],por defecto ALL
       -o <arg>   Nombre fichero de salida
       -t <arg>   Tipo de analisis a realizar[LL,SLR,LARL,LR]

C:\PLGRAM>

```

Figura E.2: *PLGRAM* sin argumento gramática.

La primera vez que se ejecuta la aplicación y se genera un fichero se crea, en el directorio donde se encuentra el ejecutable, una nueva carpeta llamada *informes*, donde se irán guardando todos los ficheros generados posteriormente.

Si solo se introducen los parámetros de la gramática y del tipo de análisis que se quiere realizar, se ejecuta el programa con el resto de valores por defecto:

- *-i* con valor *ALL*.
- *-o* que se denominará de manera genérica como el tipo de análisis elegido concatenado al nombre de la gramática seleccionada. Por ejemplo, para un análisis *LL* y una gramática *G3*, los ficheros generados se denominarán *LLG3*, cada uno con su extensión correspondiente.
- *-ca* no se generará ninguna tabla de primeros pasos de la traza de análisis.

En el supuesto de que se quiera modificar alguno de los valores que por defecto se les asignan a los parámetros anteriores, bastará con introducir en cada uno de ellos los argumentos deseados:

- `-i` se podrá elegir la extensión del fichero de salida, que será XML en el caso de querer obtener un cuestionario virtual o TEX en el caso de que se quiera para una futura impresión.
- `-o` se podrá nombrar los ficheros de salida según se desee.
- `-ca` se podrá introducir una cadena para que la aplicación genere una tabla de primeros pasos de la traza de análisis de dicha cadena. La cadena deberá introducirse entrecomillado y cada ítem separado por un espacio. Por ejemplo, `-ca "item1 item2 item3 item4"`.

Una vez introducidos todos los parámetros, se muestra la información de la gramática elegida a través de la consola.

A continuación se muestran los datos obtenidos al introducir una gramática *gramatica1.yc* y un tipo de análisis *LL*. El resto de parámetros se han dejado por defecto.

```

C:\PLGRAM>java -jar PLGRAMLineComand.jar -g gramatica1.yc -t LL
gramatica: gramatica1.yc

TERMINALES
    end,tipo,begin,codigo,id
NO TERMINALES
    A,S,C,B
SIMBOLO INICIO
    S
PRODUCCIONES
S->B A end
A->begin C
C->codigo
B->tipo
B->id B

first
C{codigo}
B{tipo id}
id{id}
A{begin}
begin{begin}
end{end}
S{tipo id}
tipo{tipo}
codigo{codigo}

follow
A{end}
S{$}
C{end}
B{begin}

Fichero.XML generado correctamente en C:\PLGRAM\informes\LLgramatica1.tex
Fichero.XML generado correctamente en C:\PLGRAM\informes\LLgramatica1.xml
C:\PLGRAM>

```

Figura E.3: *PLGRAM* con argumentos gramática y tipo de análisis.

Como se puede observar, al final de la información obtenida, la consola nos muestra que los ficheros se han generado correctamente y su ruta.

Aplicación interfaz gráfica

La interfaz gráfica pretende que el usuario tenga una visión completa y sencilla de la aplicación.

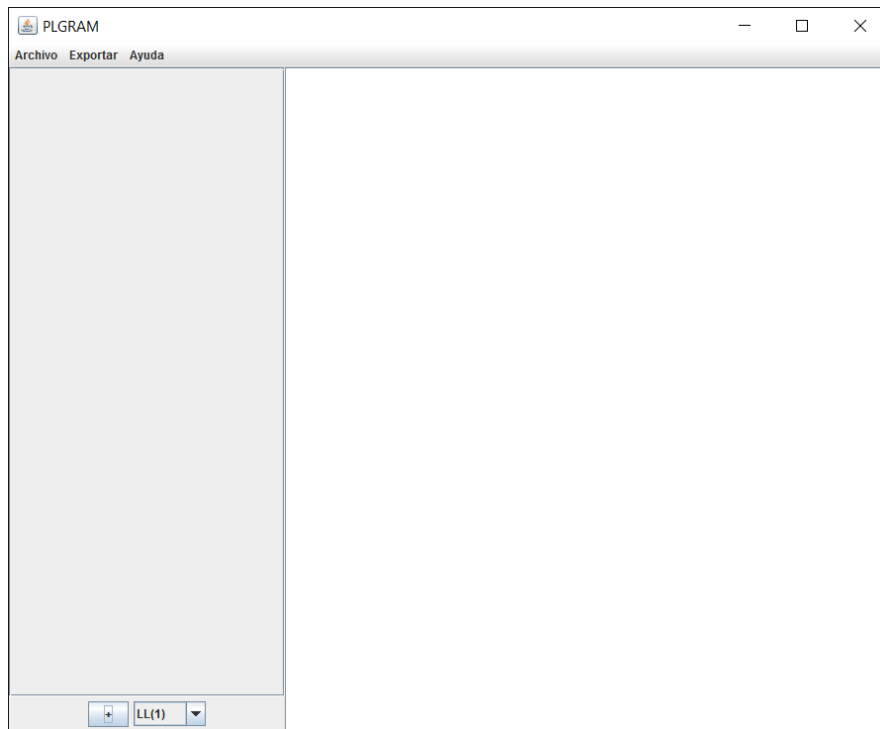


Figura E.4: Aplicación *PLGRAM*.

Añadir preguntas

Para añadir preguntas mediante la interfaz gráfica, se tienen dos opciones:

- Desde el menú Archivo. En él se ofrecen la opción de añadir preguntas utilizando los cuatro tipos de análisis.

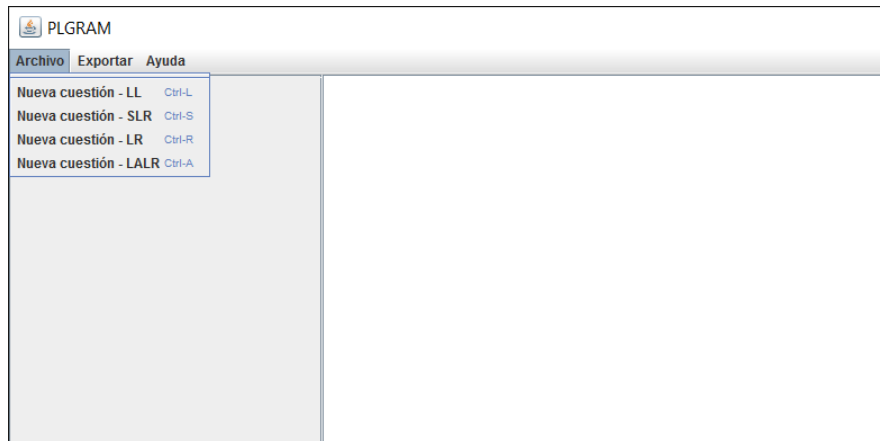


Figura E.5: Añadir preguntas desde menú Archivo.

- Utilizando el desplegable de la parte inferior de la aplicación. Al abrir dicho desplegable, se nos muestran los cuatro tipos de análisis que se pueden elegir a la hora de añadir una pregunta. Una vez seleccionado el tipo de análisis, bastaría con pulsar el botón identificado con el símbolo '+' situado a la izquierda del desplegable.

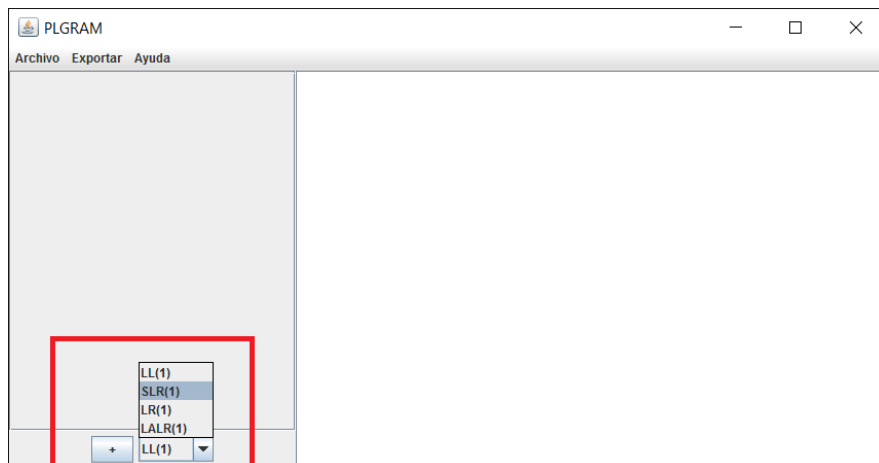


Figura E.6: Añadir preguntas desde el desplegable.

Utilizando cualquiera de las dos opciones anteriores, el resultado obtenido es una panel de preguntas que aparecerá en la parte superior izquierda de la aplicación.

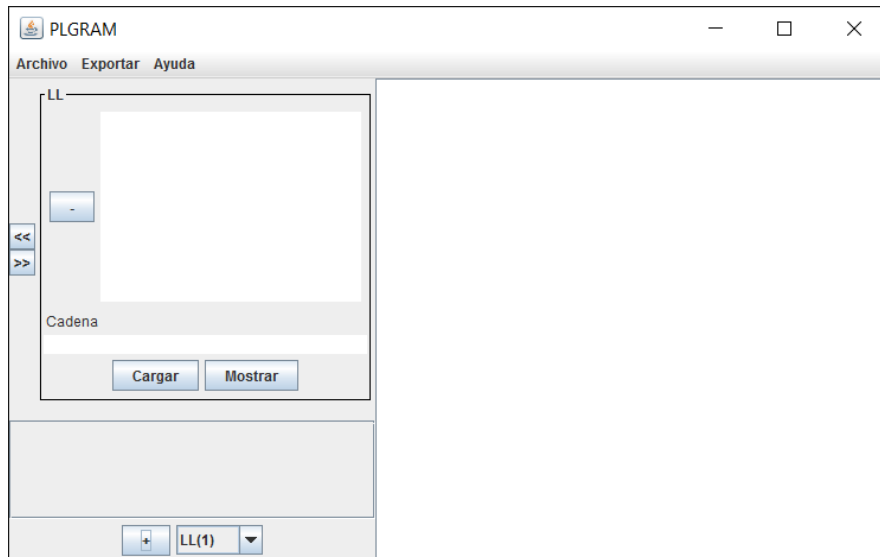


Figura E.7: Panel de preguntas.

Borrar preguntas

En el caso de querer borrar una pregunta se realiza mediante el botón situado a la izquierda del panel de preguntas identificado con el símbolo '-'.

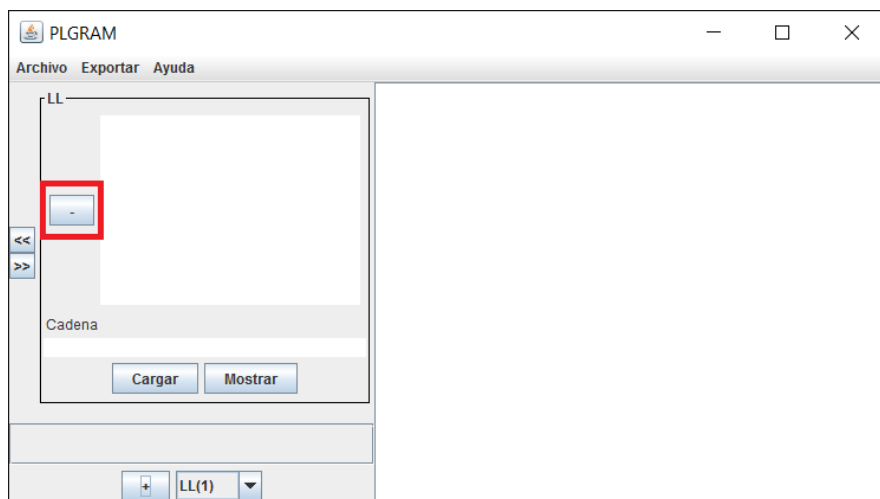


Figura E.8: Borrar preguntas.

Cambiar el orden de las preguntas

Si se tienen varias preguntas abiertas a la vez, los controles situados a la izquierda del panel de preguntas permiten modificar la posición de las preguntas hacia arriba y hacia abajo dentro de la ventana izquierda.

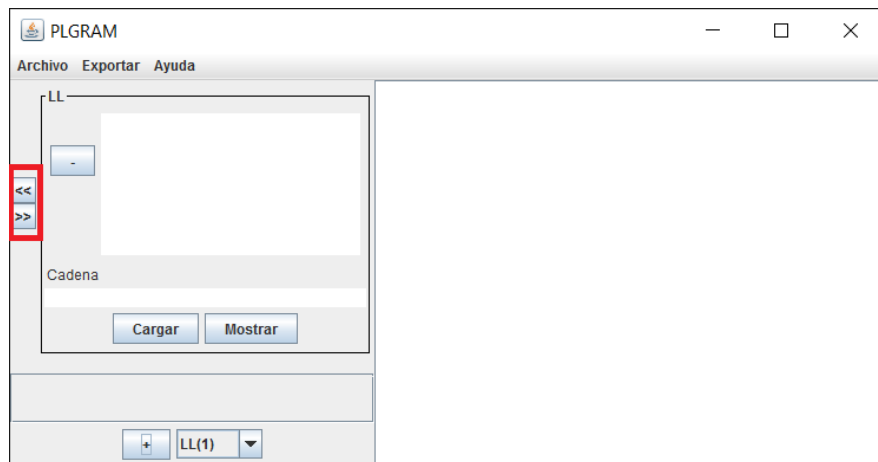


Figura E.9: Cambiar el orden de las preguntas.

Cargar gramáticas

Una vez elegido el tipo de análisis, se permite, desde el panel de preguntas, cargar la gramática deseada. Para ello, habría que pulsar el botón Cargar de dicho panel.

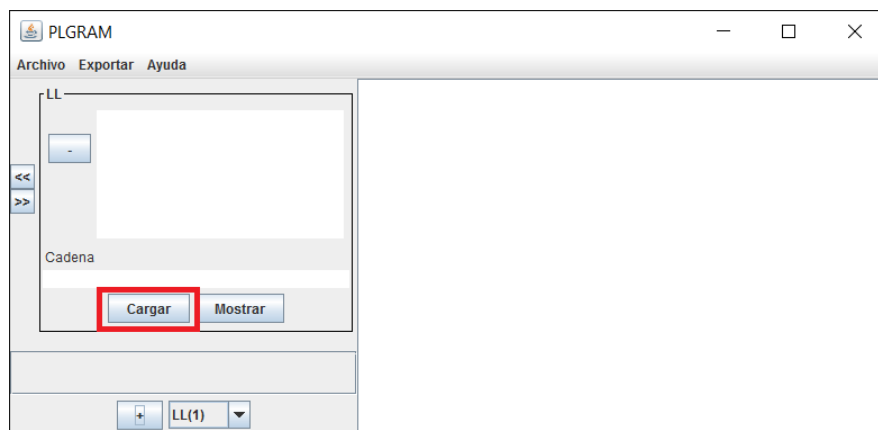


Figura E.10: Cargar gramáticas.

Aparece un explorador de ficheros en el que se puede seleccionar una de las gramáticas que se tengan guardadas.

Una vez abierta la gramática, el panel de preguntas ser rellenará con los datos de dicha gramática.

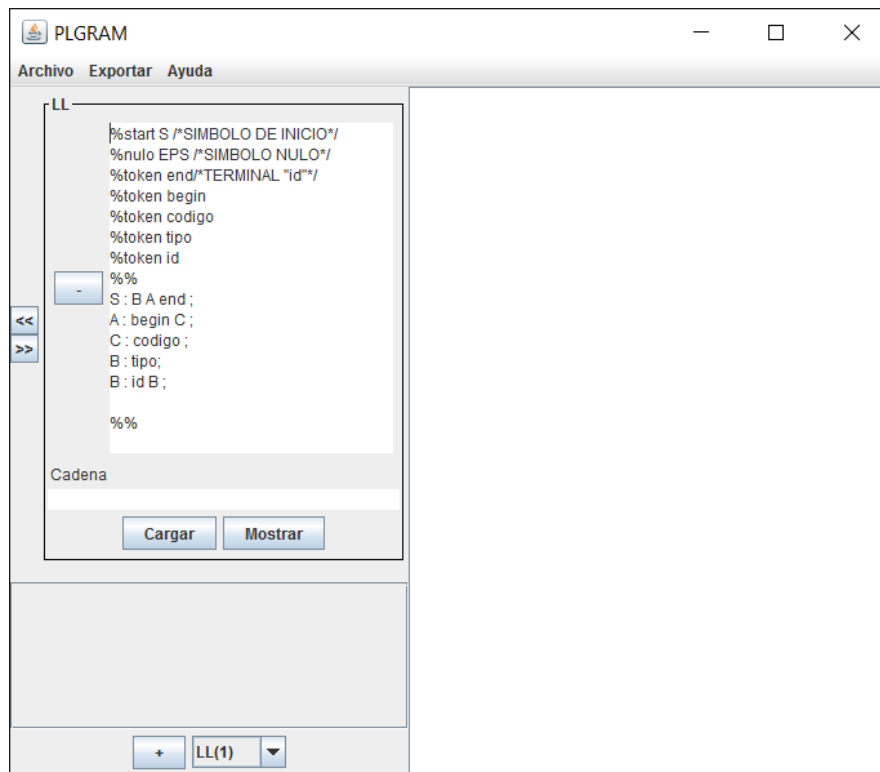


Figura E.11: Ejemplo de una gramática cargada en el panel de preguntas.

Añadir cadena

La aplicación permite añadir una cadena si se desea generar posteriormente su traza. Para ello se deberá introducir en el cuadro habilitado una cadena con los elementos separados por espacios.

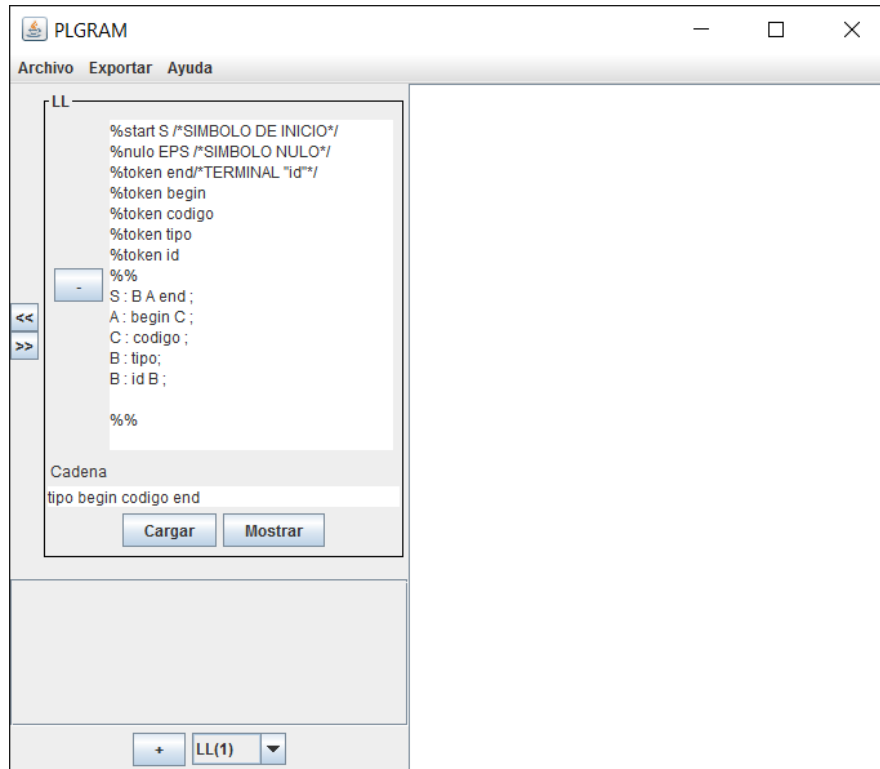


Figura E.12: Ejemplo de cadena.

Mostrar gramáticas

Elegido el tipo de análisis, la gramática deseada e introducida o no una cadena, pulsando en el botón Mostrar situado en la parte inferior del panel de preguntas, se previsualiza el resultado del análisis en la parte derecha de la pantalla.

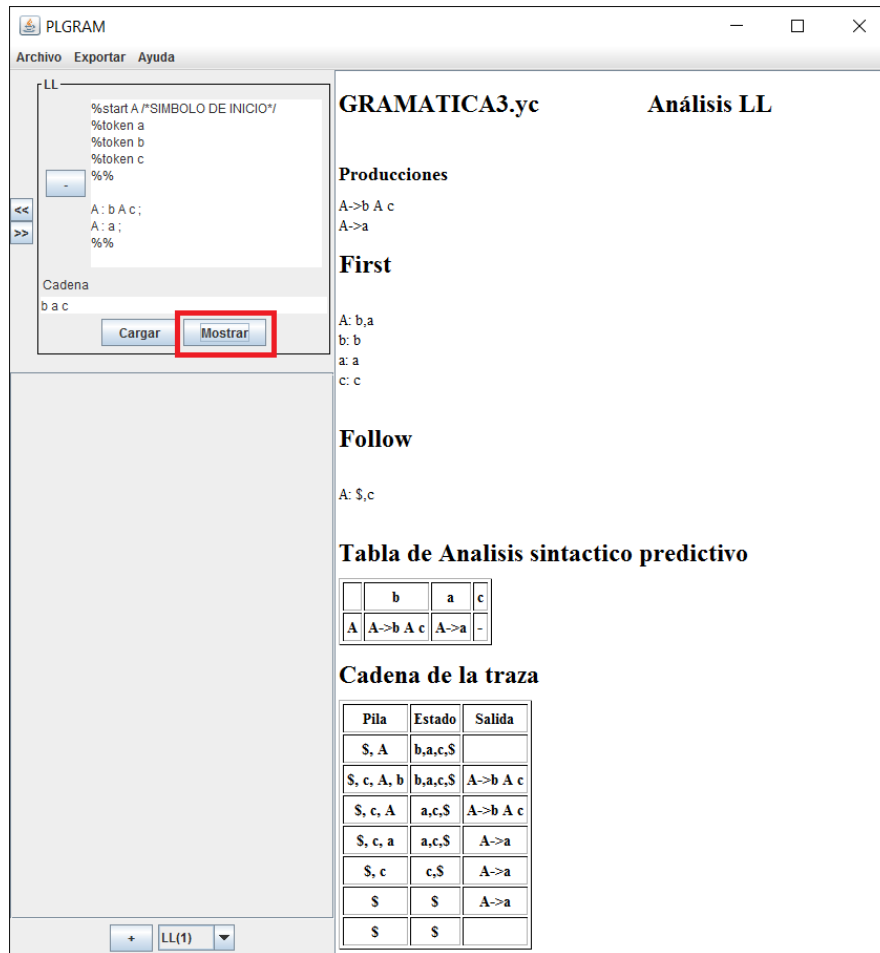


Figura E.13: Ejemplo de la previsualización de un análisis.

Cada panel de preguntas generado para cada una de las preguntas deseadas tiene su propio botón mostrar.

Exportar ficheros

Una vez obtenida la previsualización del análisis de la gramática, la aplicación permite exportar los resultados a dos tipos de ficheros: ficheros .XML y ficheros .TEX.

Para ello, debe elegirse la opción que se desee dentro del menú Exportar.

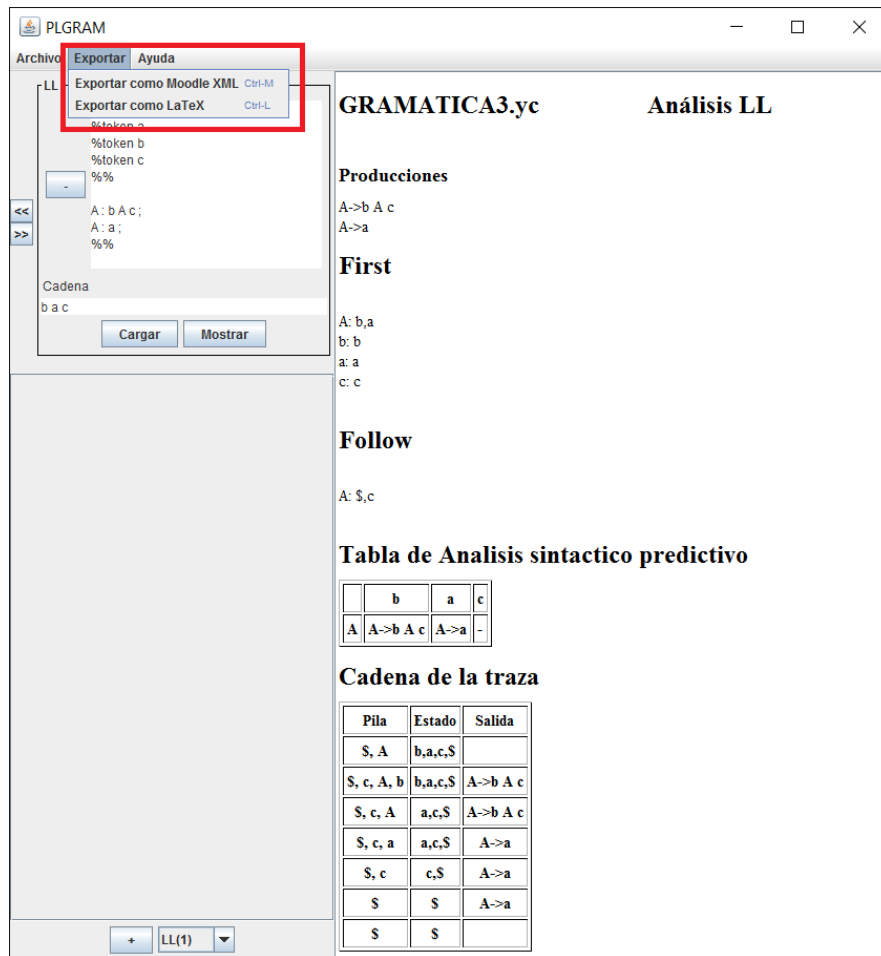


Figura E.14: Exportar ficheros.

En cualquiera de las dos opciones, aparece un cuadro de diálogo que nos informa de que el fichero se ha generado correctamente. Dicho fichero se guarda en el directorio donde se esté ejecutando la aplicación *PLGRAM.jar*.

Página web

En el menú Ayuda se encuentra la opción Página web que nos redirecciona a la página web del proyecto.

Acerca de

En el menú Ayuda se encuentra la información de la aplicación.

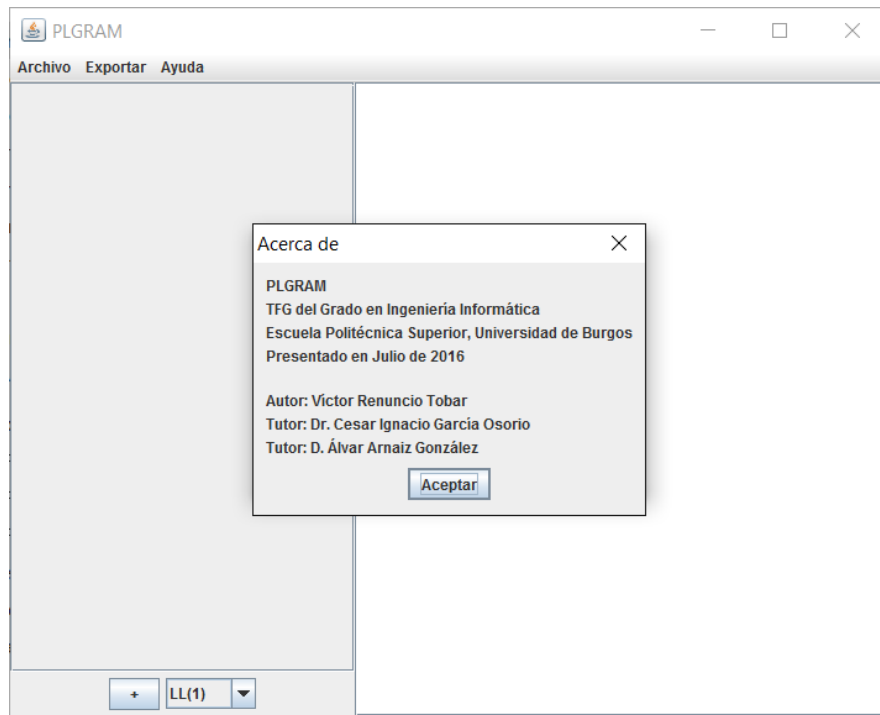


Figura E.15: Acerca de.

Ficheros obtenidos

Bien a partir de la *GUI* o mediante la *CLI* se han podido generar ficheros en los formatos .TEX o .XML

Fichero .XML

El fichero .XML obtenido tiene una estructura general común sea cual sea la gramática y el tipo de análisis realizado. Esta estructura sigue el siguiente esquema general:

```

<?xml
<quiz
  <question type=cloze
    <name
      <text
    <questiontext format=html
      <text
        <#cdata-section
    <generalfeedback format=html
      <text
    <penalty
    <hidden

```

Figura E.16: Estructura del fichero .XML.

Para poder utilizar el fichero .XML obtenido es necesario poder acceder a la plataforma de aprendizaje de Moodle.

Los pasos que se deben seguir son:

1. Ir a la página principal del curso.
2. Hacer clic en Banco de preguntas desde el bloque de Administración.
3. Pulsar en Importar archivos.
4. Seleccionar la opción de formato XML Moodle.
5. Desde la siguiente ventana pulsar en Examinar.
6. Seleccionar el fichero .XML en nuestro ordenador.
7. Pulsar en Importar este archivo.

Los puntos 5 y 6 se pueden realizar arrastrando el archivo al recuadro que se muestra.

A la hora de importar el fichero .XML hay que tener en cuenta lo siguiente:

- Los nombres de los archivos y carpetas no deben tener caracteres especiales como palabras acentuadas, tabulaciones, ñ, retornos de carro, espacios en blanco, ni símbolos de sistema como / o : , además no es recomendable combinar mayúsculas y minúsculas en los nombres.
- El tamaño del archivo no debe superar el límite permitido que se muestra en la ventana.

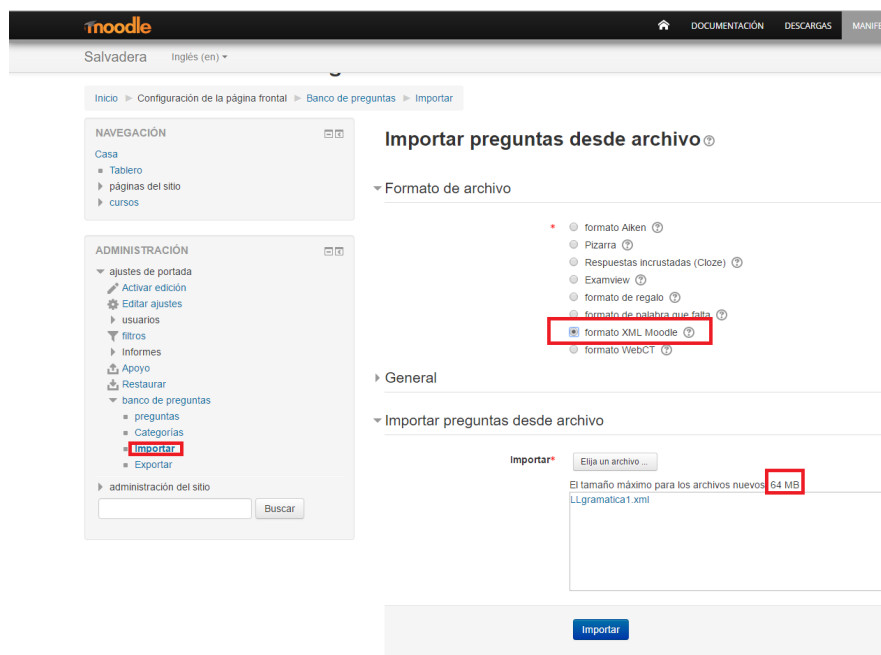


Figura E.17: Importar fichero .XML en Moodle.

Una vez importado el fichero .XML deseado, aparece una primera previusualización de los datos que se obtendrán a continuación. Basta con dar a continuar para acceder a la siguiente pantalla.

En ella, aparecen todos los ficheros que se hayan importado. Para visualizar uno de ellos, se hace click en la lupa del archivo correspondiente.

Question bank

Select a category:
 Default for Front page (3) ▼

The default category for questions shared in context 'Front page':

☐ Show question text in the question list

[Search options](#) ▼

☒ Also show questions from subcategories

☐ Also show old questions

[Create a new question ...](#)

Question	Created by First name / Surname / Date	Last modified by First name / Surname / Date
LLgramatica1	Admin User 1 July 2016, 1:47 AM	Admin User 1 July 2016, 1:47 AM
LRgramatica1	Admin User 1 July 2016, 1:47 AM	Admin User 1 July 2016, 1:47 AM
SLRgramatica2	Admin User 1 July 2016, 1:46 AM	Admin User 1 July 2016, 1:48 AM

With selected:

[Delete](#) [Move to >>](#) [Default for Front page \(3\)](#) ▼

Figura E.18: Lista de ficheros .XML en Moodle.

Un ejemplo de cuestionario obtenido sería el siguiente:

Preview question: LLgramatica3

Question 1
 Answer saved
 Marked out of 19.00

Completar las tablas que siguen, dada la siguiente gramática (A es el axioma, y se usa el símbolo S para representar la cadena vacía):

$A \rightarrow b A c$
 $A \rightarrow a$

Tabla de los conjuntos FIRST y FOLLOW

	FIRST	FOLLOW
A	b, a ▼	S, c ▼

Tabla de análisis sintáctico predictivo

	b	a	c
A	$A \rightarrow b A c$ ▼	$A \rightarrow a$ ▼	— ▼

Traza de análisis para la cadena "b a c S"

PILA	ENTRADA	SALIDA
S, A	b, a, c, S	
S, c, A, b	b, a, c, S	$A \rightarrow b A c$ ▼
S, c, A	a, c, S	$A \rightarrow b A c$ ▼
S, c, a	a, c, S	$A \rightarrow a$ ▼
S, c	c, S	$A \rightarrow a$ ▼
S	S	

Figura E.19: Ejemplo de cuestionario LL resuelto en Moodle.

Preview question: LRgramatica3

Question 1
Answer saved
Marked out of 65.00

Completar los conjuntos de ítems LR(1), las tablas LR(1) y la traza de análisis de la cadena "b a c \$", para la gramática que sigue (A es el axioma, se usa el símbolo \$ para representar la cadena vacía):

$A \rightarrow b A c$
 $A \rightarrow a$

Conjuntos de ítems marcados En la operación de lectura, la lectura de símbolos se hace en el orden [A, b, a, c]. En las operaciones de cierre las producciones con el ítem al principio se añaden en el mismo orden en que aparecen en la gramática.

Conjunto 0	Conjunto 1	Conjunto 2	Conjunto 3
Ítem marcado	Ítem marcado	Ítem marcado	Ítem marcado
Simbolos de anticipación	Simbolos de anticipación	Simbolos de anticipación	Simbolos de anticipación
$A \rightarrow b A c$	$A \rightarrow b A c$	$A \rightarrow b A c$	$A \rightarrow b A c$
$A \rightarrow b A c$	$A \rightarrow b A c$	$A \rightarrow b A c$	$A \rightarrow b A c$
$A \rightarrow b A c$	$A \rightarrow b A c$	$A \rightarrow b A c$	$A \rightarrow b A c$

Conjunto 4	Conjunto 5	Conjunto 6	Conjunto 7
Ítem marcado	Ítem marcado	Ítem marcado	Ítem marcado
Simbolos de anticipación	Simbolos de anticipación	Simbolos de anticipación	Simbolos de anticipación
$A \rightarrow b A c$	$A \rightarrow b A c$	$A \rightarrow b A c$	$A \rightarrow b A c$
$A \rightarrow b A c$	$A \rightarrow b A c$	$A \rightarrow b A c$	$A \rightarrow b A c$
$A \rightarrow b A c$	$A \rightarrow b A c$	$A \rightarrow b A c$	$A \rightarrow b A c$

Conjunto 8	Conjunto 9
Ítem marcado	Ítem marcado
Simbolos de anticipación	Simbolos de anticipación
$A \rightarrow b A c$	$A \rightarrow b A c$
$A \rightarrow b A c$	$A \rightarrow b A c$
$A \rightarrow b A c$	$A \rightarrow b A c$

Figura E.20: Ejemplo de cuestionario $LR(1)$ resuelto en Moodle. Conjunto de ítems

Tabla ACCIÓN e IR A

Para especificar las acciones se utiliza una codificación similar a la vista en clase:

- dj: significa desplazar e ir al estado j.
- rj: significa reducir utilizando la producción j-ésima.
- ACP: representa la acción de aceptar la cadena.
- : se utiliza para indicar que la casilla está vacía (no tiene acción).
- En la parte de acción se especifica únicamente el estado al que pasa el autómata.

Tabla de ACCIÓN					Tabla de IR A
estado	b	a	c	\$	A
0	d2	d3	-	-	1
1	-	-	-	ACP	-
2	d5	d6	-	-	4
3	-	-	-	r3	-
4	-	-	d7	-	-
5	d5	d6	-	-	8
6	-	-	r3	-	-
7	-	-	-	r2	-
8	-	-	d9	-	-
9	-	-	r2	-	-

Figura E.21: Ejemplo de cuestionario $LR(1)$ resuelto en Moodle. Tabla de ACCIÓN e IR A

Fichero .TEX

Al tratarse de un fichero .TEX se requiere un editor de textos adaptado a L^AT_EX para utilizarlo.

Una vez abierto el fichero, su tratamiento es muy sencillo. Bastará con compilar el código y se obtendrá directamente un visionado del cuestionario que se ha realizado.

Se muestra un ejemplo de un fichero .TEX abierto con T_EXMaKer. De manera general, se puede observar el código a la izquierda y del documento generado a la derecha.

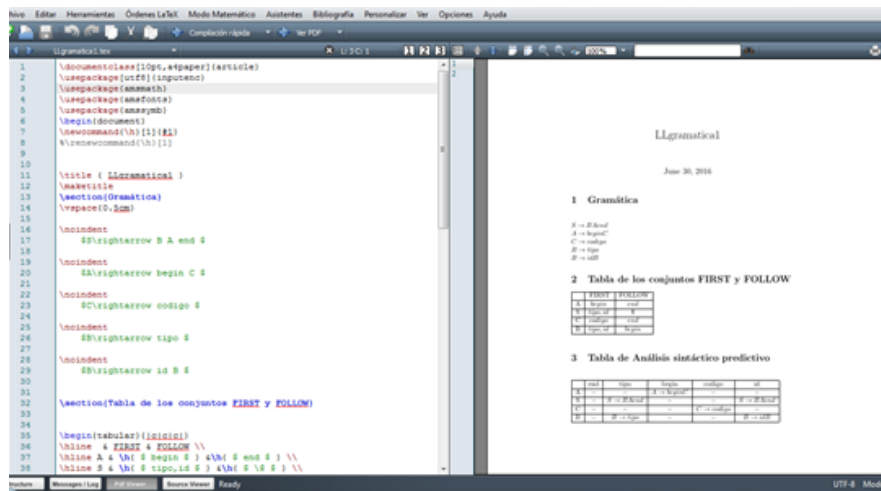


Figura E.22: Ejemplo fichero .TEX para un análisis LL.

Para una mejor comprensión, se muestra un detalle de cómo compilar el código para obtener el documento. Para ello, basta con hacer click en las flechas resaltadas en la parte superior de la pantalla del editor.

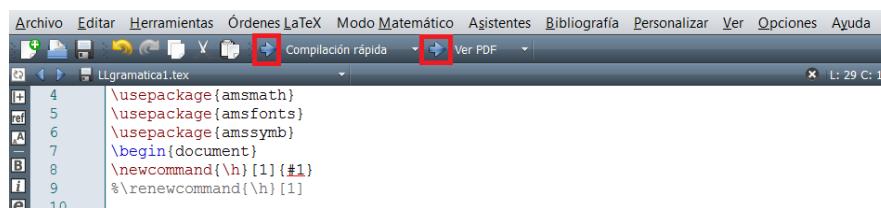


Figura E.23: Compilación de un fichero .TEX.

Es necesario destacar que para cada tipo de análisis elegido, se genera un fichero .TEX diferente siguiendo los requisitos de cada una de ellas.

La aplicación ofrece dos opciones en lo que a los ficheros .TEX respecta. Así, se podrá obtener un documento en el que el cuestionario esté completo

con las respuestas correctas y otro documento en el que el cuestionario se encuentre vacío, pensado para ser respondido por un tercero.

- Obtención del cuestionario completo. Es la opción que se genera por defecto y por tanto no será necesario tratar el código para su obtención. De cualquier manera, debería estar activa la siguiente línea:

```
\newcommand{\h}[1]{#1}
```

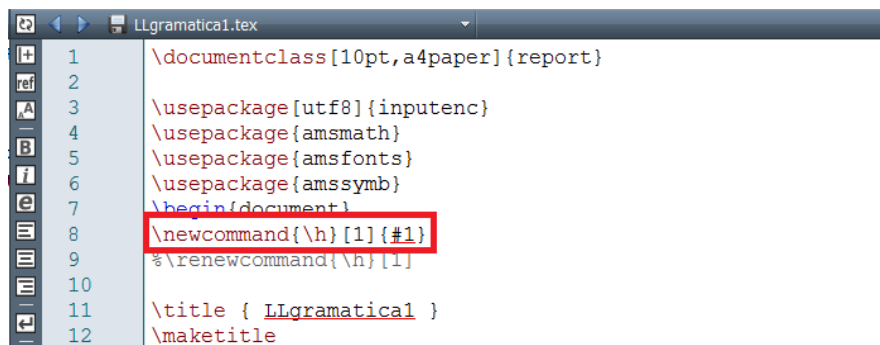


Figura E.24: Obtención del cuestionario completo.

- Obtención del cuestionario en blanco. Para ello, bastaría con eliminar el símbolo % de delante de la siguiente línea:

```
%\renewcommand{\h}[1]
```

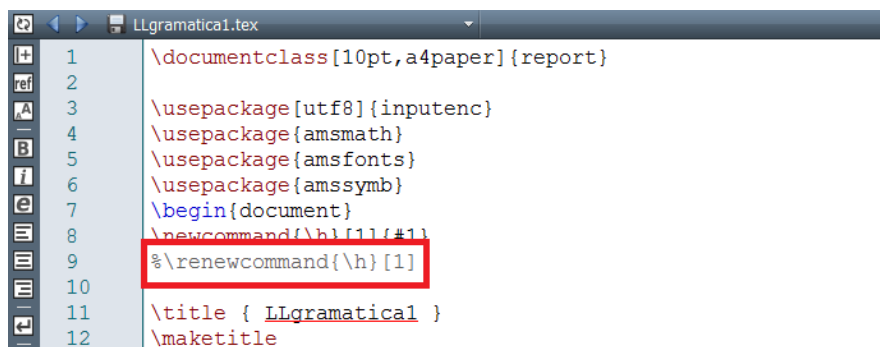
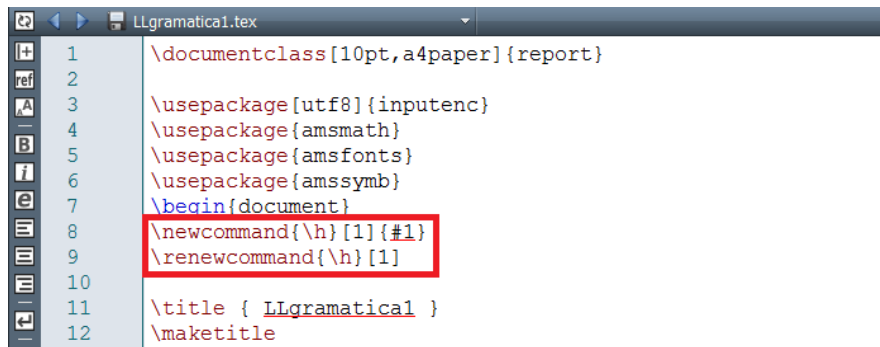


Figura E.25: Obtención del cuestionario en blanco: descomentar.

Así, ambas líneas deberían estar activas.



```
1 \documentclass[10pt,a4paper]{report}
2
3 \usepackage[utf8]{inputenc}
4 \usepackage{amsmath}
5 \usepackage{amsfonts}
6 \usepackage{amssymb}
7 \begin{document}
8 \newcommand{\h}[1]{#1}
9 \renewcommand{\h}[1]
10
11 \title{ LLgramatica1 }
12 \maketitle
```

Figura E.26: Obtención del cuestionario en blanco.

Siempre que se compile código, se crea un archivo .PDF en el mismo directorio donde se encuentra el archivo .TEX, con los cuestionarios elegidos.

Bibliografía

- [1] Requisitos de java. <https://www.java.com/es/download/help/sysreq.xml>.
- [2] Apache License. Version 2.0. <http://www.apache.org/licenses/>, 2004.
- [3] Steffen Macke. Dia diagram editor, jun 1991. Dia es un software gratuito disponible bajo los términos de la GPLv2.
- [4] UML ObjectAid. Explorer for eclipse. <http://www.objectaid.com/>, 2014.